Capstone Project: Pneumonia Detection

Final Report

Author: Sivasankar Vennala

V1.0

Table of Contents

1. Project Description

In this capstone project, the goal is to build a pneumonia detection system, to locate the position of inflammation in an image. Tissues with sparse material, such as lungs which are full of air, do not absorb the X-rays and appear black in the image. Dense tissues such as bones absorb X-rays and appear white in the image. While we are theoretically detecting “lung opacities”, there are lung opacities that are not pneumonia related. In the data, some of these are labelled “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.

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.

Details about the data and dataset files are given in below link :

https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data

The objectives of the project are,

- Learn to how to do build an Object Detection Model
- Use transfer learning to fine-tune a model
- Learn to set the optimizers, loss functions, epochs, learning rate, batch size, check-pointing, early stopping etc.
- Read different research papers of given domain to obtain the knowledge of advanced models for the given problem.

2. Introduction

A. Defining problem statement

The objective is to explore the dataset for RSNA Pneumonia Detection 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.

We start by exploring the DICOM data, we extract the meta information from the DICOM files and visualize the various features of the DICOM images, grouped by age, sex.

We will use data from URL: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data

B. Need of the study/project

Diagnosing pneumonia requires review of a chest radiograph (CXR) by highly trained specialists and confirmation through clinical history, vital signs and laboratory exams. However, the diagnosis of pneumonia on CXR is complicated because of a number of other conditions in the lungs such as fluid overload (pulmonary edema), bleeding, volume loss (atelectasis or collapse), lung cancer, or post-radiation or surgical changes. Outside of the lungs, fluid in the pleural space (pleural effusion) also appears as increased opacity on CXR.

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. When available, comparison of CXRs of the patient taken at different time points and correlation with clinical symptoms and history are helpful in making the diagnosis.

C. Understanding business/social opportunity

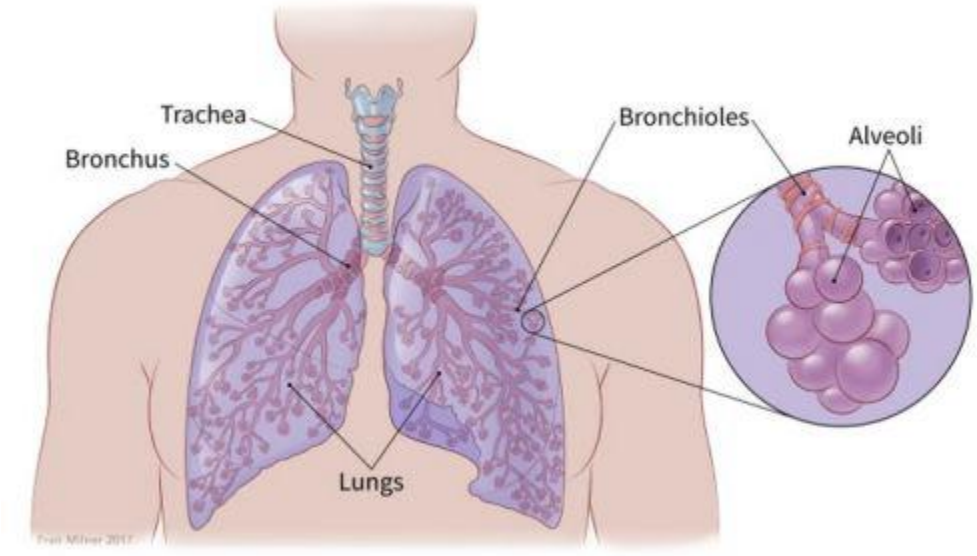
Automating Pneumonia screening in chest radiographs, providing affected area details through a bounding box. Assist physicians to make better clinical decisions or even replace human judgement in certain functional areas of healthcare (ex. radiology). Guided by relevant clinical questions, powerful AI techniques can unlock clinically relevant information hidden in the massive amount of data, which in turn can assist clinical decision making.

“Spotting the signs is not easy… Health workers, who often have low education levels, also lack the required knowledge to recognise pneumonia in time.” --Dr Camielle Noordam

“Detecting pneumonia in chest radiography can be difficult for radiologists. The appearance of pneumonia in X-ray images is often vague… discrepancies cause considerable variability among radiologists in the diagnosis of pneumonia” --Dr Mark I. Neuman

3. Preliminary Research on Data

Pneumonia is a disease that is caused by bacterial or viral infection of the lungs. This causes the air sacs to fill up with fluid and substantially affects breathing.



One of the ways to diagnose pneumonia is to analyse the Chest X-Ray (further on CXR). The dataset used by this project is mainly a part of a competition called RSNA Pneumonia Detection Challenge organized by Radiological Society of North America. The dataset contains separate set of Chest X-ray image for training and test. The training data is provided as a set of patient Ids and bounding boxes. Bounding boxes are defined as follows: x-min, y min, width, height. There is also a binary target column, Target, indicating pneumonia or non-pneumonia. There may be multiple rows per patient Id. All provided images are in DICOM format.

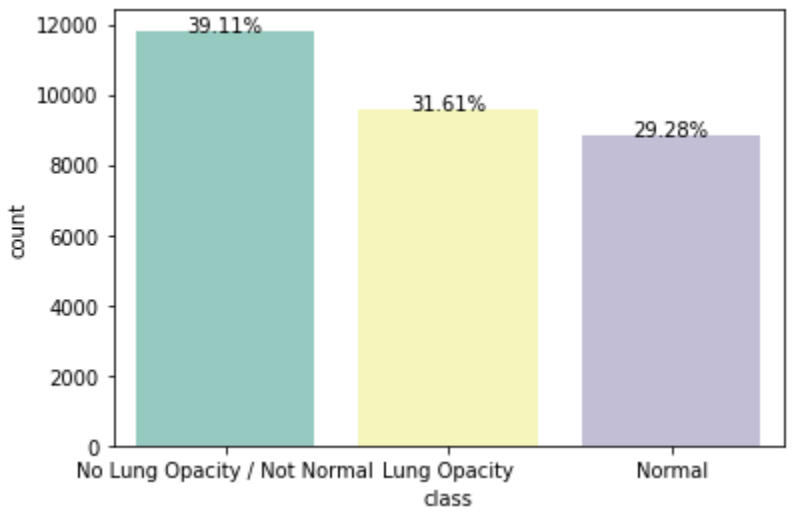
DICOM – Digital Imaging and Communications in Medicine is known as an international standard for medical images and everything that is related to them. DICOM images are known to have high quality, since the diagnosis requires as clear information as possible. Nowadays, this format is implemented in almost all medical domains like radiology, cardiology, radiotherapy devices, ophthalmology and even dentistry. When DICOM was introduced, it has revolutionized radiology, allowing practitioners to switch from X-Ray films to digital format.

Lets see what data DICOM images provided to us

```
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: "1"
(0020, 0013) Instance Number                     IS: "1"
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.14300000000000002, 0.14300000000000002]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 142006 elements
```

From Figure, you can see the specific data inside a DICOM file provided to us by the radiological community in the USA. It contains several important bits of information, such as the unique patient ID, the view position of the body when the scan was taken, the gender and age of the patient. All this information can be used to further explore the possible solutions to the problem.

Dataset which we used in the project contains 3 different classes which are No Lung Opacity / Not Normal, Lung Opacity and Normal.

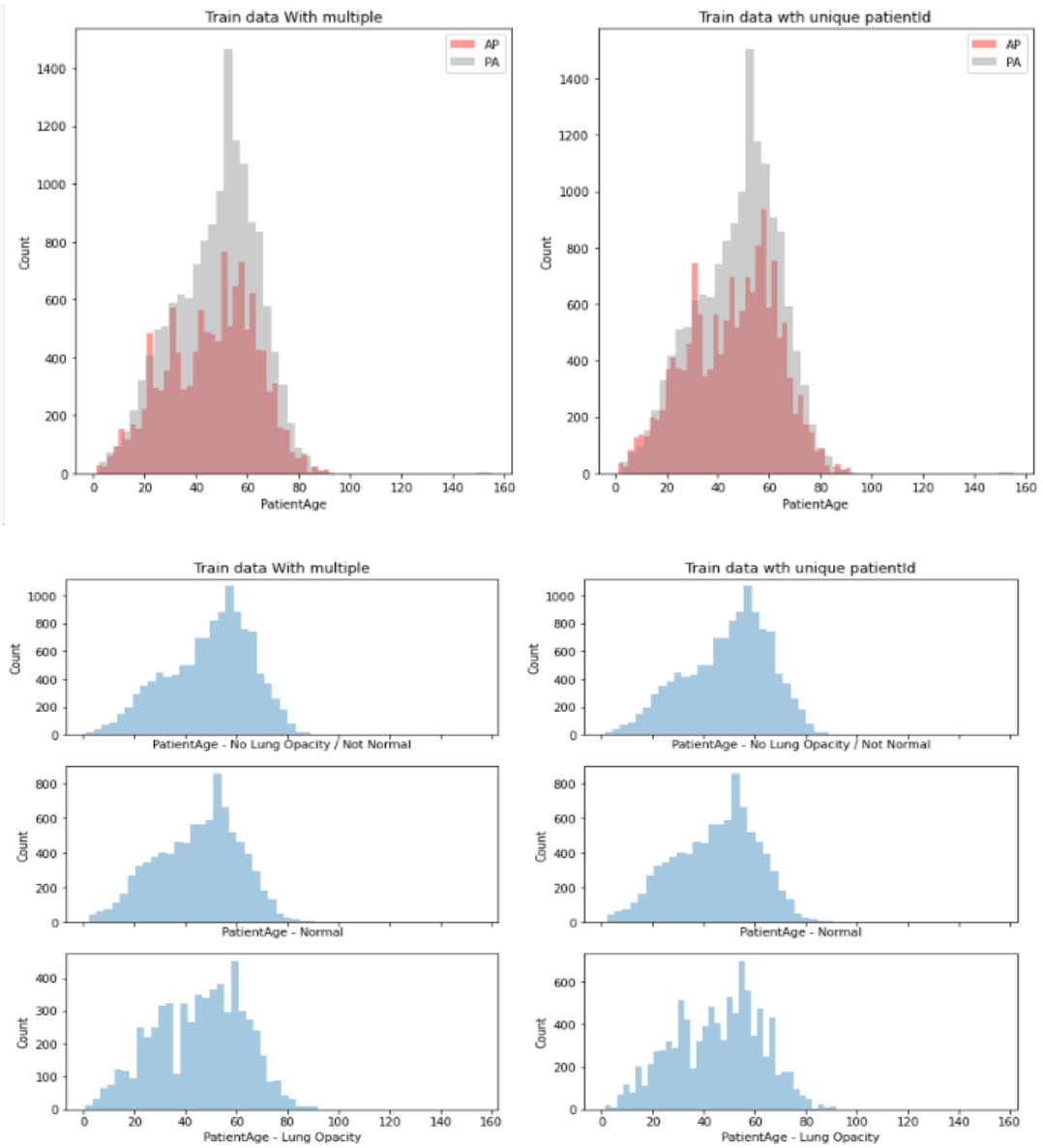


4. Methodology

A. Exploratory Data Analysis

1. Data Pre-processing

The most important part of data science and machine learning is data and data must be neat and clean for models to process it. Data pre-processing methodologies vary with regards to the data types. When it comes to image data, there are other preparation methods. Every state-of-the-art model was built with some assumptions about the data like the size or the number of channels. So, the dataset needs to be resized, or the architecture of the network changed. The image can be resized to get rid of unnecessary information in the background. When these steps are done, data needs to be analysed using various statistical methods to get some insights. Usually scatter plots and histograms help a lot.



2. Data Exploration

Now, let us dive deeper into the data. As it has been said earlier, the data comes in DICOM format. The image below is a part of a DICOM file.

we can see above sample picture of a patient's with pneumonia. Knowing how to read CXR helps in understanding the data better. Let us break down the process of X-Ray imaging, X-Ray passes through the body and reaches the detector on the other side. During its journey it encounters matter with different densities, dense materials like bones and tissues absorb the radiation and appear white on CXR, other materials like air do not really absorb X-Ray, thus it reaches the target.

As we can see in above sample DICOM Images air is indicated by the black colour, bones are the white colour, tissues and fluids are grey. Now, we can analyse the pictures above, as we can see the sample pictures 1 has pneumonia which is labelled by the bounding boxes, and sample pictures 2 has clear and healthy lungs. In addition to pneumonia, similar opaque regions can be produced by other dense objects, like lung cancer or fluid like water in the lungs. Considering this, there are have 3 classes in the dataset: Lung Opacity, Not Normal and Normal. The target variable can have 2 values, 0 and 1. So it is a binary classification task. However, the positions of bounding boxes must be predicted as well.

5. Methods and models

The main goal is predicting whether pneumonia exists in a given image. They do so by predicting bounding boxes around areas of the lung. Samples without bounding boxes are negative and contain no definitive evidence of pneumonia. Samples with bounding boxes indicate evidence of pneumonia. When making predictions, there should predict as many bounding boxes as necessary, in the format: confidence x-min, y-min, width height. We use different kind of Convolutional Neural Network Based model to solve this this problem. They are as follow,

A. Customised Basic CNN approach

Initially as part of this Capstone project we developed a customised basic CNN model and the performance of the customized CNN model relies on sequential layered CNN architecture for the task of classifying normal and positive cases. The dataset were randomly divided into 80% for training and 20% for validation testing and there is a separate dataset for testing. Images are down-sampled to 224×224 pixel resolutions.

Each CNN block has a convolutional layer with Relu layer followed by a batch normalization and Max pooling layer. Padding is added to the convolutional layers to ensure that the spatial output dimensions match the original input. The first and second convolutional layer has 64 filters each and the third has 128 filters and so on. A filter size of 3×3 is used uniformly across the layers.

The search ranges include customised learning rate, mean_iou and iou_loss parameters, Adam optimiser parameters respectively. A Sigmoid classifier follows the up sampling to output the prediction distribution between the normal and abnormal images. The convergence of the model is attributed to proper weight initializations. A method for weight initializations for networks with ReLU non-linear activations is used.

Model Block:

```
def create_network(input_size, channels):
    inputs = keras.Input(shape=(input_size, input_size, 1))
    x = Conv2D(32, 3, activation='relu', padding='same')(inputs)

    x = BatchNormalization(momentum=0.9)(x)
    x = Conv2D(64, 1, activation='relu', padding='same')(x)
    x = MaxPool2D(2)(x)

    x = BatchNormalization(momentum=0.9)(x)
    x = Conv2D(64, 3, activation='relu', padding='same', use_bias=False)(x)
    x = BatchNormalization(momentum=0.9)(x)
    x = Conv2D(64, 3, activation='relu', padding='same', use_bias=False)(x)

    x = BatchNormalization(momentum=0.9)(x)
    x = Conv2D(64, 3, activation='relu', padding='same', use_bias=False)(x)
    x = BatchNormalization(momentum=0.9)(x)
```

```
x = Conv2D(64, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(128, 1, activation='relu', padding='same')(x)
x = MaxPool2D(2)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(128, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(128, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(128, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(128, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(256, 1, activation='relu', padding='same')(x)
x = MaxPool2D(2)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(256, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(256, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(256, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(256, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(512, 1, activation='relu', padding='same')(x)
x = MaxPool2D(2)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(512, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(512, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(512, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(512, 3, activation='relu', padding='same', use_bias=False)(x)


x = BatchNormalization(momentum=0.9)(x)
```

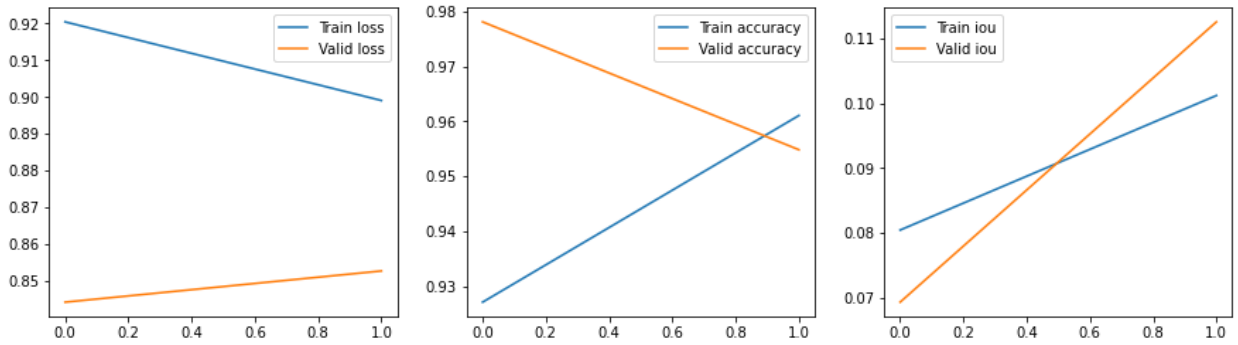
```
x = Conv2D(1024, 1, activation='relu', padding='same')(x)
x = MaxPool2D(2)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(1024, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(1024, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(1024, 3, activation='relu', padding='same', use_bias=False)(x)
x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(1024, 3, activation='relu', padding='same', use_bias=False)(x)

x = BatchNormalization(momentum=0.9)(x)
x = Conv2D(1, 1, activation='sigmoid')(x)
outputs = UpSampling2D(32)(x)
model = Model(inputs=inputs, outputs=outputs)
return model
```

Model Summary:

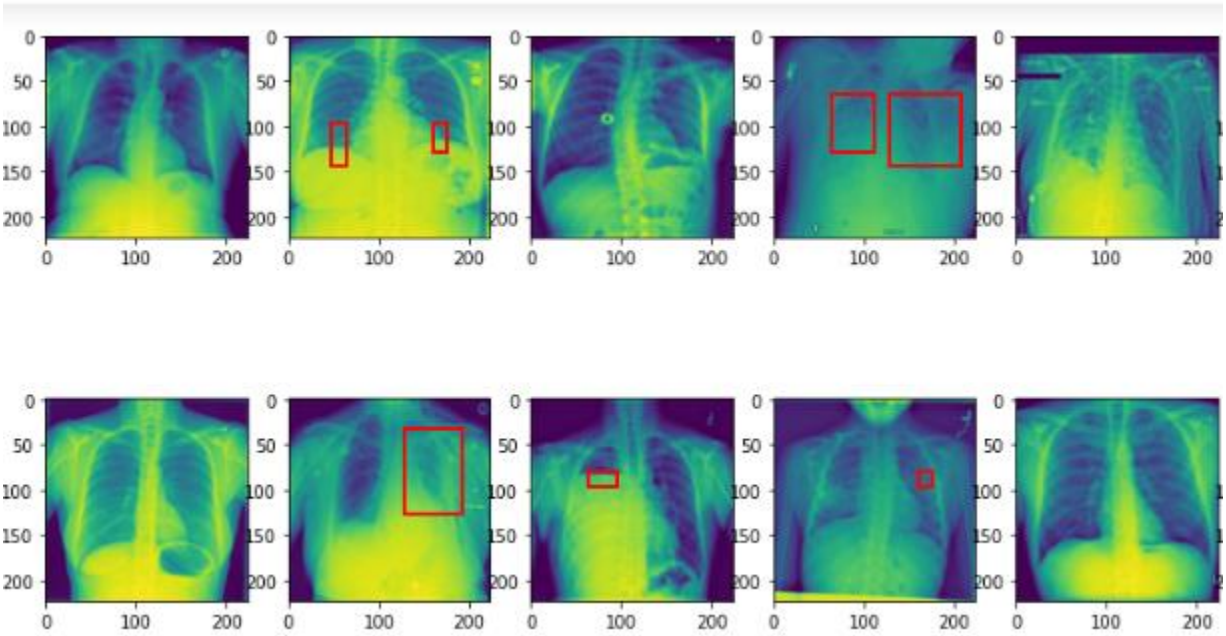
```
=================================================================
Total params: 12,728,961
Trainable params: 12,719,297
Non-trainable params: 9,664
_________________________________________________________________
```

Overall customised basic CNN approach given a accuracy: 0.9271 and val_accuracy: 0.9781 with 5 Epochs and batch size of 64. if we can train the model more epochs we can get more good accuracy. But due to computational limitations we cannot able to train more than 5 epochs, still its given good accuracy.

Metrics:



Predictions:



2.Improvements:

Initially we decided to play with customised basic CNN model by changing different hyper parameters and add/removing different layers. As part of these changes we introduced a LeakyReLU layer instead of Relu layer to the model and image size down sample to 256x256 and played with changing different learning rates using below customised cosine function.

```
def cosine_annealing(x):
    lr = 0.001
    epochs = 5
    return lr*(np.cos(np.pi*x/epochs)+1.)/2
learning_rate = tf.keras.callbacks.LearningRateScheduler(cosine_annealing)
```

Model Block:

```
def create_sample(channels, inputs):
    x = keras.layers.BatchNormalization(momentum=0.9)(inputs)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 1, padding='same', use_bias=False)(x)
    x = keras.layers.MaxPool2D(2)(x)
    return x

def create_resblock(channels, inputs):
    x = keras.layers.BatchNormalization(momentum=0.9)(inputs)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
```

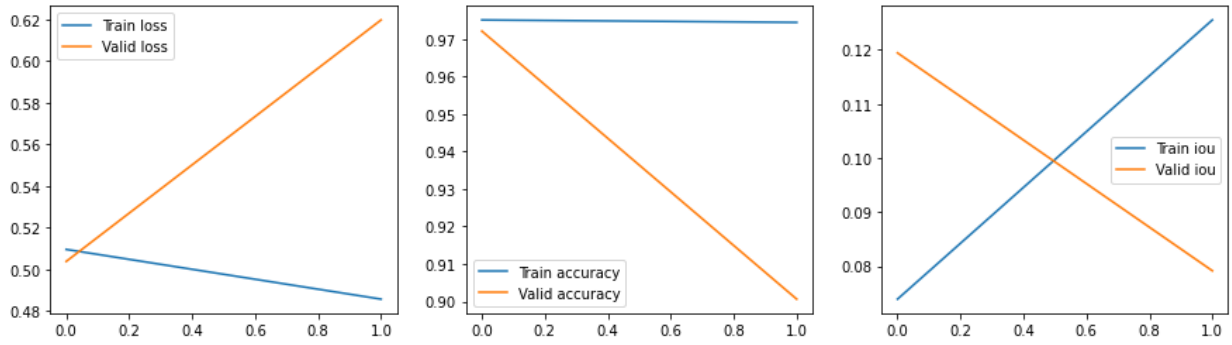
```
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(x)
    return keras.layers.add([x, inputs])

def create_network(input_size, channels, n_blocks=2, depth=4):
    # input
    inputs = keras.Input(shape=(input_size, input_size, 1))
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(inputs)
    # residual blocks
    for d in range(depth):
        channels = channels * 2
        x = create_sample(channels, x)
        for b in range(n_blocks):
            x = create_resblock(channels, x)
    # output
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(1, 1, activation='sigmoid')(x)
    outputs = keras.layers.UpSampling2D(2**depth)(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
```

Model Summary:

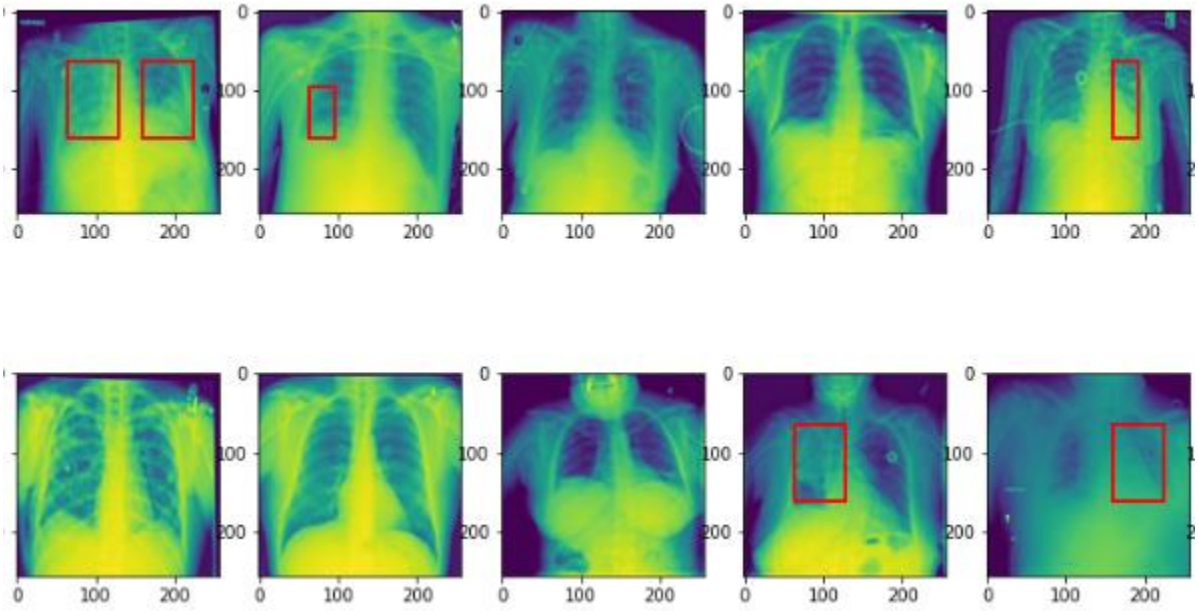
```
=================================================================
Total params: 51,021,985
Trainable params: 51,002,081
Non-trainable params: 19,904
_________________________________________________________________
_________________________
```

We observed that Basic CNN model with small changes like activation function and different learning rates given very good accuracy: 0.9751 and val_accuracy: 0.9721 with 5 Epochs with batch size of 32 and image size of 256x256, if we can train the model more epochs we can get more good accuracy. But due to computational limitations we cannot able to train more than 5 epochs, still its given good accuracy.

Metrics:



Predictions:



Further we decide to try using pretrained models like MobileNet, VGG, RCN, FCN etc and other segmentation techniques to see the results and performance.

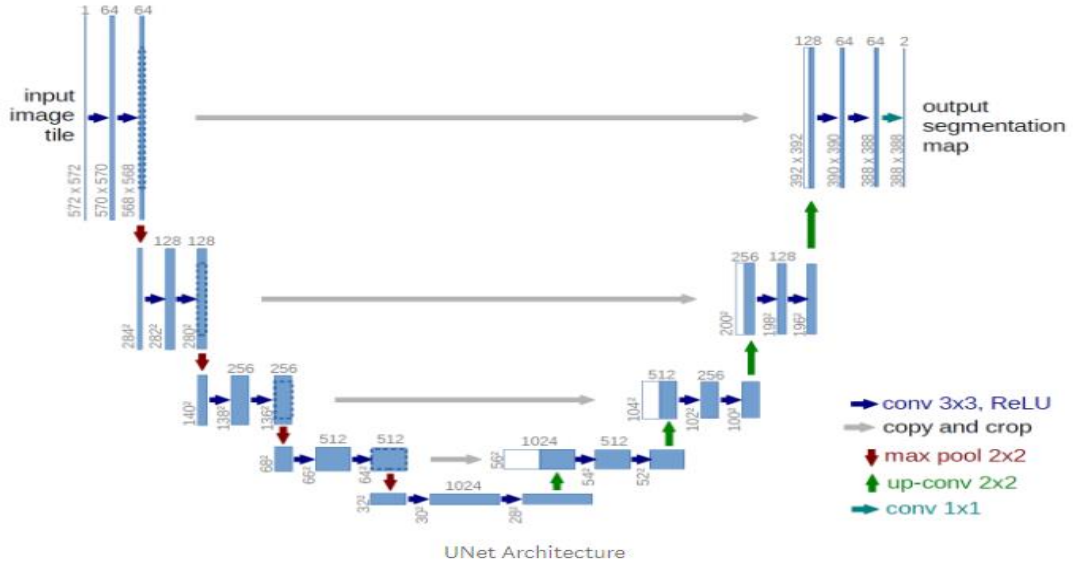
B. Pre-trained models used

We used following pre trained model further to check the model performance and further improvements.

1. **UNET Model**
2. **VGG16**
3. **InceptionV3**
4. **Resnet50 (using Matterport MASK_RCNN)**

UNET Model:

The architecture looks like a 'U' which justifies its name. This architecture consists of three sections: The contraction, The bottleneck, and the expansion section. The contraction section is made of many contraction blocks. Each block takes an input applies two 3X3 convolution layers followed by a 2X2 max pooling. The number of kernels or feature maps after each block doubles so that architecture can learn the complex structures effectively. The bottommost layer mediates between the contraction layer and the expansion layer. It uses two 3X3 CNN layers followed by 2X2 up convolution layer.



UNet Architecture

But the heart of this architecture lies in the expansion section. Similar to contraction layer, it also consists of several expansion blocks. Each block passes the input to two 3X3 CNN layers followed by a 2X2 up sampling layer. Also after each block number of feature maps used by convolutional layer get half to maintain symmetry. However, every time the input is also get appended by feature maps of the corresponding contraction layer. This action would ensure that the features that are learned while contracting the image will be used to reconstruct it. The number of expansion blocks is as same as the number of contraction block. After that, the resultant mapping passes through another 3X3 CNN layer with the number of feature maps equal to the number of segments desired.

The dataset were randomly divided into 80% for training and 20% for validation testing and there is a separate dataset for testing. Images are down-sampled to 224×224 pixel resolutions. The search ranges include customised learning rate, mean_iou and iou_loss parameters, Adam optimiser parameters respectively. For training purpose we used batch size of 64 with 5 Epochs.

Model Block:

```
def conv2d_block(input_tensor, n_filters, kernel_size=3, batchnorm=True):
    # first layer
    x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), kernel_initializer="he_normal",
               padding="same")(input_tensor)
```

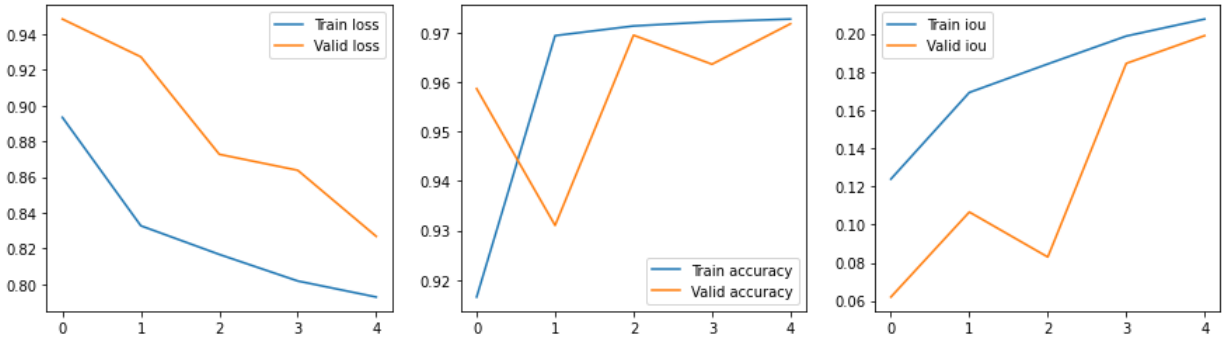
```
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation("relu")(x)
    # second layer
    x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), kernel_initializer="he_normal",
               padding="same")(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation("relu")(x)
    return x

def create_model(input_img, n_filters=32, dropout=0.5, batchnorm=True):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*1, kernel_size=3, batchnorm=batchnorm)
    p1 = MaxPooling2D((2, 2)) (c1)
    p1 = Dropout(dropout*0.5)(p1)

    c2 = conv2d_block(p1, n_filters=n_filters*2, kernel_size=3, batchnorm=batchnorm)
    p2 = MaxPooling2D((2, 2)) (c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters=n_filters*4, kernel_size=3, batchnorm=batchnorm)
    p3 = MaxPooling2D((2, 2)) (c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters=n_filters*8, kernel_size=3, batchnorm=batchnorm)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)
    p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters=n_filters*16, kernel_size=3, batchnorm=batchnorm)

    # expansive path
    u6 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c5)
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters=n_filters*8, kernel_size=3, batchnorm=batchnorm)

    u7 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters=n_filters*4, kernel_size=3, batchnorm=batchnorm)

    u8 = Conv2DTranspose(n_filters*2, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters=n_filters*2, kernel_size=3, batchnorm=batchnorm)
```

```
u9 = Conv2DTranspose(n_filters*1, (3, 3), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
u9 = Dropout(dropout)(u9)
c9 = conv2d_block(u9, n_filters=n_filters*1, kernel_size=3, batchnorm=batchnorm)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
model = Model(inputs=[input_img], outputs=[outputs])
return model
```

Model Summary:

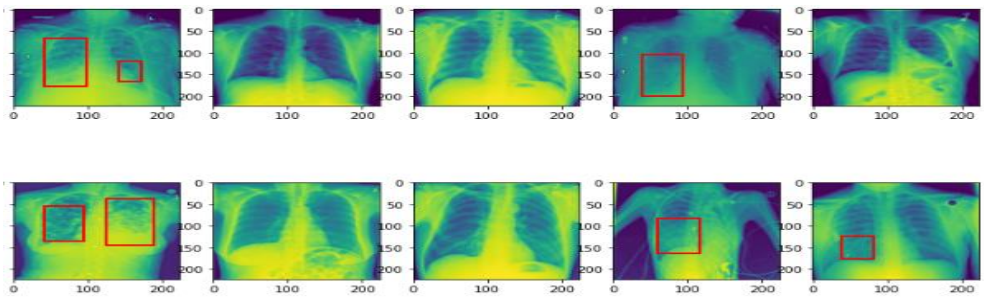
```
==================================================================================================
Total params: 8,641,697
Trainable params: 8,635,809
Non-trainable params: 5,888
__________________________________________________________________________________________________
```

Metrics:



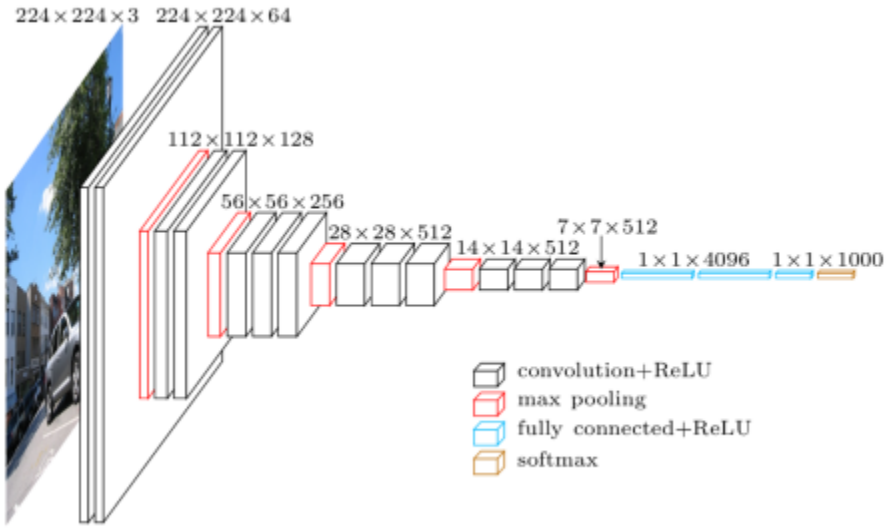
We can see UNET model given accuracy: 0.9728 and val_accuracy: 0.9719 with 5 Epochs, if we can train the model more epochs we can get more good accuracy. But due to computational limitations we cannot able to train more than 5 epochs, still its given good accuracy.

Predictions:



VGG16 Pretrained Model:

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.



The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time.

The dataset were randomly divided into 75% for training and 25% for validation testing and there is a separate dataset for testing. Images are down-sampled to 384×384 pixel resolutions. For training purpose we used batch size of 24 with 20 Epochs.

Model Block:

```
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten, Input, Conv2D, multiply, LocallyConnected2D, Lambda, AvgPool2D
from keras.models import Model
from keras.optimizers import Adam
from keras import layers
pt_features = Input(base_pretrained_model.get_output_shape_at(0)[1:], name = 'feature_input')
pt_depth = base_pretrained_model.get_output_shape_at(0)[-1]
from keras.layers import BatchNormalization
bn_features = BatchNormalization()(pt_features)
gap = GlobalAveragePooling2D()(bn_features)

gap_dr = Dropout(DROPOUT)(gap)
dr_steps = Dropout(DROPOUT)(Dense(DENSE_COUNT, activation = 'linear', use_bias=False)(gap_dr))
dr_steps = BatchNormalization()(dr_steps)
dr_steps = layers.LeakyReLU(0.1)(dr_steps)
out_layer = Dense(t_y.shape[1], activation = 'softmax')(dr_steps)

attn_model = Model(inputs = [pt_features],
                   outputs = [out_layer], name = 'trained_model')

attn_model.summary()

from keras.models import Sequential
from keras.optimizers import Adam
pneu_model = Sequential(name = 'combined_model')
base_pretrained_model.trainable = False
pneu_model.add(base_pretrained_model)
pneu_model.add(attn_model)
pneu_model.compile(optimizer = Adam(lr = LEARN_RATE), loss = 'categorical_crossentropy',
                   metrics = ['categorical_accuracy'])
pneu_model.summary()
```

Model Summary:

Customised layers will be added to make it better suited for the classification problem.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
feature_input (InputLayer)   (None, 12, 12, 512)       0
_________________________________________________________________
```

batch_normalization_1 (Batch (None, 12, 12, 512) 2048

_____

global_average_pooling2d_1 ((None, 512) 0

_____

dropout_1 (Dropout) (None, 512) 0

_____

dense_1 (Dense) (None, 128) 65536

_____

dropout_2 (Dropout) (None, 128) 0

_____

batch_normalization_2 (Batch (None, 128) 512

_____

leaky_re_lu_1 (LeakyReLU) (None, 128) 0

_____

dense_2 (Dense) (None, 3) 387

==========

Total params: 68,483

Trainable params: 67,203

Non-trainable params: 1,280

_____

VGG Model:

_____

Layer (type) Output Shape Param #

==========

vgg16 (Model) (None, 12, 12, 512) 14714688

_____

trained_model (Model) (None, 3) 68483

==========

Total params: 14,783,171

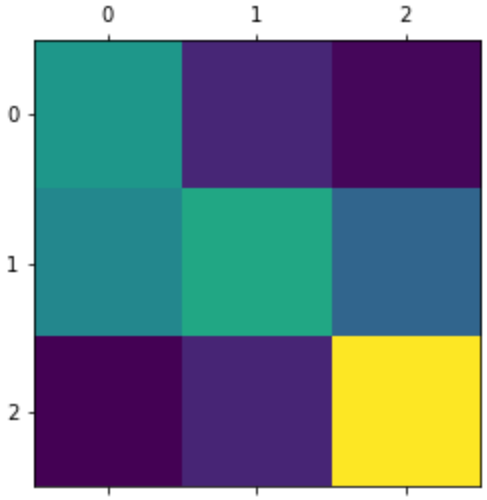
Trainable params: 67,203

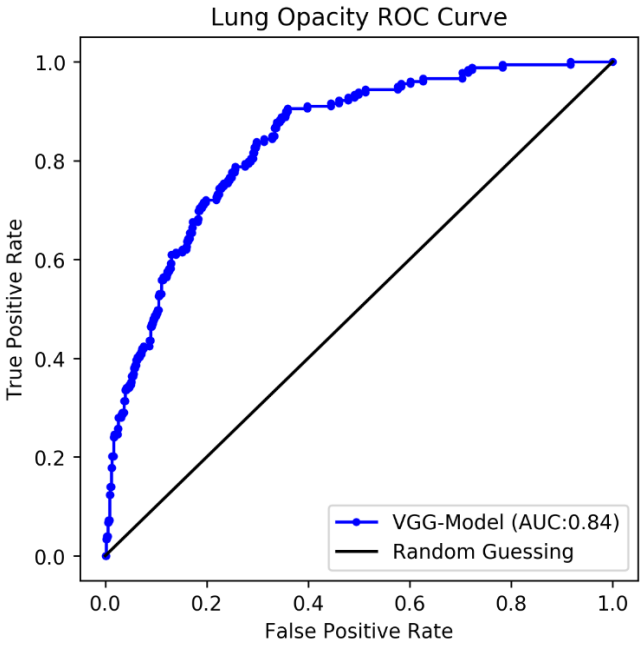
Non-trainable params: 14,715,968

_____

Metrics:

	precision	recall	f1-score	support
Lung Opacity	0.51	0.72	0.60	179
No Lung Opacity / Not Normal	0.68	0.42	0.52	342
Normal	0.70	0.84	0.76	279
avg / total	0.65	0.63	0.62	800

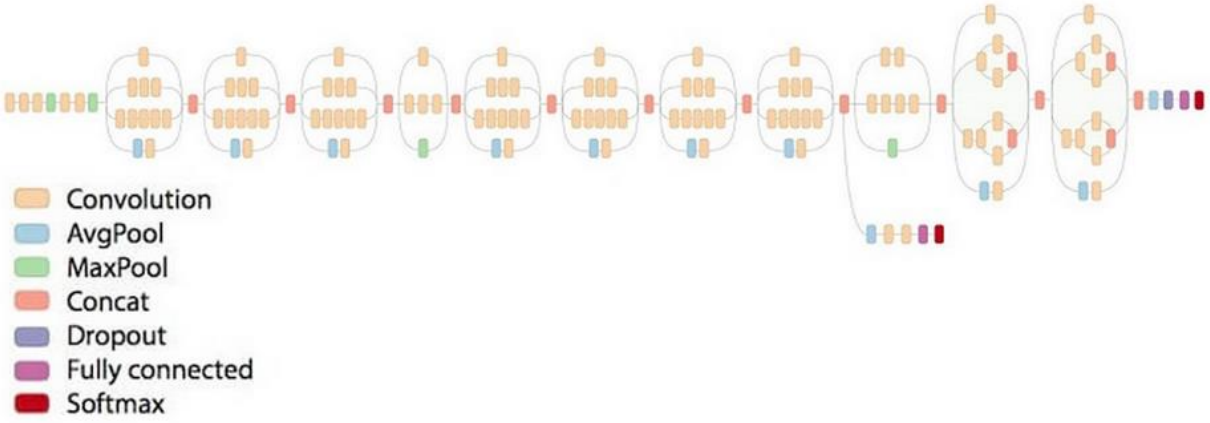




VGG16 model given accuracy: 0.84 and val_accuracy: 0.6337 with 20 Epochs, which are under fitting model results.

Inception V3 Pretrained Model:

Inception V3 by Google is the 3rd version in a series of Deep Learning Convolutional Architectures. Inception V3 was trained using a dataset of 1,000 classes from the original ImageNet dataset which was trained with over 1 million training images, the Tensorflow version has 1,001 classes which is due to an additional background class not used in the original ImageNet. Inception V3 was trained for the ImageNet Large Visual Recognition Challenge where it was a first runner up.



The dataset were randomly divided into 75% for training and 25% for validation testing and there is a separate dataset for testing. Images are down-sampled to 224×224 pixel resolutions. For training purpose we used batch size of 24 with 20 Epochs.

Model Block:

```
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten, Input, Conv2D, multiply, LocallyConnected2D, Lambda, AvgPool2D
from keras.models import Model
from keras.optimizers import Adam
from keras import layers
pt_features = Input(base_pretrained_model.get_output_shape_at(0)[1:], name = 'feature_input')
pt_depth = base_pretrained_model.get_output_shape_at(0)[-1]
from keras.layers import BatchNormalization
bn_features = BatchNormalization()(pt_features)
gap = GlobalAveragePooling2D()(bn_features)

gap_dr = Dropout(DROPOUT)(gap)
dr_steps = Dropout(DROPOUT)(Dense(DENSE_COUNT, activation = 'linear', use_bias=False)(gap_dr))
dr_steps = BatchNormalization()(dr_steps)
dr_steps = layers.LeakyReLU(0.1)(dr_steps)
out_layer = Dense(t_y.shape[1], activation = 'softmax')(dr_steps)

attn_model = Model(inputs = [pt_features],
                   outputs = [out_layer], name = 'trained_model')
```

```
attn_model.summary()

from keras.models import Sequential
from keras.optimizers import Adam
pneu_model = Sequential(name = 'combined_model')
base_pretrained_model.trainable = False
pneu_model.add(base_pretrained_model)
pneu_model.add(attn_model)
#pneu_model.compile(optimizer = Adam(lr = LEARN_RATE), loss = 'categorical_crossentropy',
#                   metrics = ['categorical_accuracy'])
pneu_model.compile(optimizer = Adam(lr = LEARN_RATE), loss = iou_loss,
                   metrics = [mean_iou,'accuracy'])
pneu_model.summary()
```

Model Summary:

Customised layers will be added to make it better suited for the classification problem.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
feature_input (InputLayer)   (None, 5, 5, 2048)        0
_________________________________________________________________
batch_normalization_95 (Batc (None, 5, 5, 2048)        8192
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               262144
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
batch_normalization_96 (Batc (None, 128)               512
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 387
=================================================================
Total params: 271,235
Trainable params: 266,883
Non-trainable params: 4,352
_________________________________________________________________
```

Inception V3 Layer:

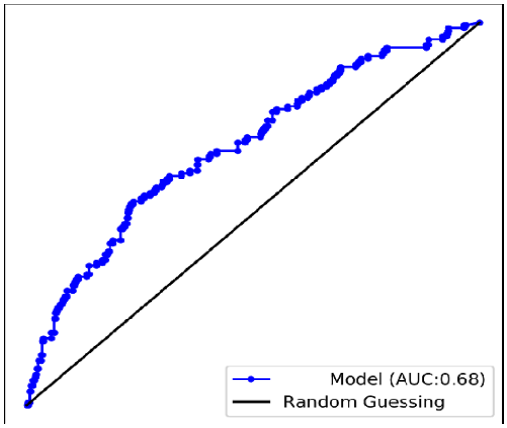
Layer (type)	Output Shape	Param #
inception_v3 (Model)	(None, 5, 5, 2048)	21802784
trained_model (Model)	(None, 3)	271235

Total params: 22,074,019
Trainable params: 266,883
Non-trainable params: 21,807,136

Metrics:

	precision	recall	f1-score	support
Lung Opacity	0.00	0.00	0.00	137
No Lung Opacity / Not Normal	0.56	0.67	0.61	266
Normal	0.57	0.81	0.67	197
avg / total	0.43	0.56	0.49	600

ROC Curve:



Inception V3 model given accuracy: 0.68 and val_accuracy: 0.5517 with 20 Epochs, which also a under fitting model results.

Lets try segmentation model:

Resnet50: (using MASK_RCNN)

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 a image or a video. You give it a 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.

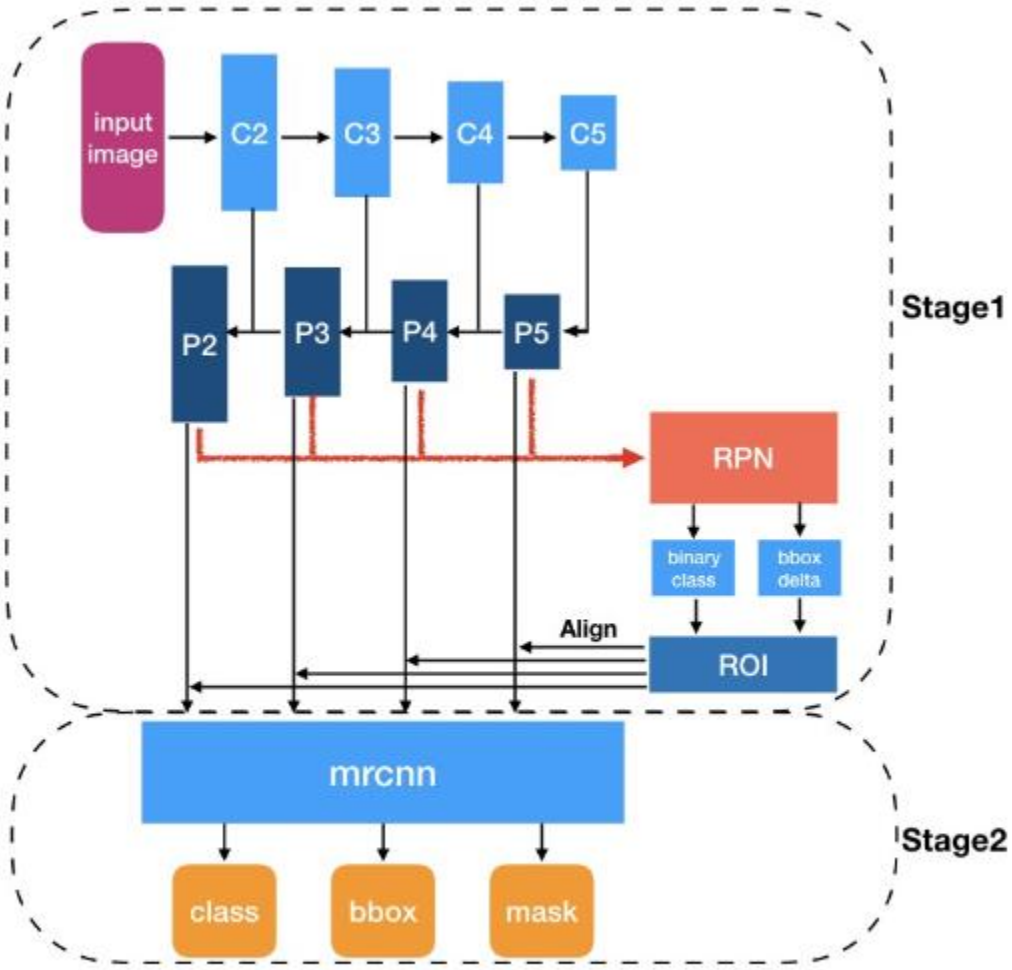
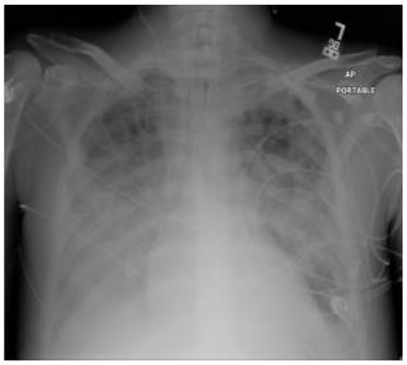
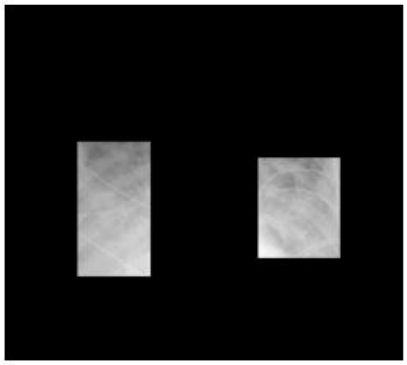


Illustration of Mask RCNN structure

We used Matterport's MASK_RCNN model to train, basically this model used 2 backbone network like Resnet50 and Resnet101. We used Resnet50 as backbone network for our model and image size of 256x256. We used 200 epocs to train backbone network and different augmentation techniques.

Sample DICOM image with bounding box:

Selecting layers to train:

```
conv1                  (Conv2D)
bn_conv1               (BatchNorm)
res2a_branch2a         (Conv2D)
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)
bn3b_branch2c          (BatchNorm)
```

```
res3c_branch2a          (Conv2D)
bn3c_branch2a           (BatchNorm)
res3c_branch2b          (Conv2D)
bn3c_branch2b           (BatchNorm)
res3c_branch2c          (Conv2D)
bn3c_branch2c           (BatchNorm)
res3d_branch2a          (Conv2D)
bn3d_branch2a           (BatchNorm)
res3d_branch2b          (Conv2D)
bn3d_branch2b           (BatchNorm)
res3d_branch2c          (Conv2D)
bn3d_branch2c           (BatchNorm)
res4a_branch2a          (Conv2D)
bn4a_branch2a           (BatchNorm)
res4a_branch2b          (Conv2D)
bn4a_branch2b           (BatchNorm)
res4a_branch2c          (Conv2D)
res4a_branch1           (Conv2D)
bn4a_branch2c           (BatchNorm)
bn4a_branch1            (BatchNorm)
res4b_branch2a          (Conv2D)
bn4b_branch2a           (BatchNorm)
res4b_branch2b          (Conv2D)
bn4b_branch2b           (BatchNorm)
res4b_branch2c          (Conv2D)
bn4b_branch2c           (BatchNorm)
res4c_branch2a          (Conv2D)
bn4c_branch2a           (BatchNorm)
res4c_branch2b          (Conv2D)
bn4c_branch2b           (BatchNorm)
res4c_branch2c          (Conv2D)
bn4c_branch2c           (BatchNorm)
res4d_branch2a          (Conv2D)
bn4d_branch2a           (BatchNorm)
res4d_branch2b          (Conv2D)
bn4d_branch2b           (BatchNorm)
res4d_branch2c          (Conv2D)
bn4d_branch2c           (BatchNorm)
res4e_branch2a          (Conv2D)
bn4e_branch2a           (BatchNorm)
res4e_branch2b          (Conv2D)
bn4e_branch2b           (BatchNorm)
res4e_branch2c          (Conv2D)
bn4e_branch2c           (BatchNorm)
res4f_branch2a          (Conv2D)
bn4f_branch2a           (BatchNorm)
res4f_branch2b          (Conv2D)
bn4f_branch2b           (BatchNorm)
res4f_branch2c          (Conv2D)
bn4f_branch2c           (BatchNorm)
res5a_branch2a          (Conv2D)
bn5a_branch2a           (BatchNorm)
res5a_branch2b          (Conv2D)
bn5a_branch2b           (BatchNorm)
```

```
res5a_branch2c          (Conv2D)
res5a_branch1           (Conv2D)
bn5a_branch2c           (BatchNorm)
bn5a_branch1            (BatchNorm)
res5b_branch2a          (Conv2D)
bn5b_branch2a           (BatchNorm)
res5b_branch2b          (Conv2D)
bn5b_branch2b           (BatchNorm)
res5b_branch2c          (Conv2D)
bn5b_branch2c           (BatchNorm)
res5c_branch2a          (Conv2D)
bn5c_branch2a           (BatchNorm)
res5c_branch2b          (Conv2D)
bn5c_branch2b           (BatchNorm)
res5c_branch2c          (Conv2D)
bn5c_branch2c           (BatchNorm)
fpn_c5p5                (Conv2D)
fpn_c4p4                (Conv2D)
fpn_c3p3                (Conv2D)
fpn_c2p2                (Conv2D)
fpn_p5                  (Conv2D)
fpn_p2                  (Conv2D)
fpn_p3                  (Conv2D)
fpn_p4                  (Conv2D)
In model:  rpn_model
    rpn_conv_shared        (Conv2D)
    rpn_class_raw          (Conv2D)
    rpn_bbox_pred          (Conv2D)
mrcnn_mask_conv1        (TimeDistributed)
mrcnn_mask_bn1          (TimeDistributed)
mrcnn_mask_conv2        (TimeDistributed)
mrcnn_mask_bn2          (TimeDistributed)
mrcnn_class_conv1       (TimeDistributed)
mrcnn_class_bn1         (TimeDistributed)
mrcnn_mask_conv3        (TimeDistributed)
mrcnn_mask_bn3          (TimeDistributed)
mrcnn_class_conv2       (TimeDistributed)
mrcnn_class_bn2         (TimeDistributed)
mrcnn_mask_conv4        (TimeDistributed)
mrcnn_mask_bn4          (TimeDistributed)
mrcnn_bbox_fc           (TimeDistributed)
mrcnn_mask_deconv       (TimeDistributed)
mrcnn_class_logits      (TimeDistributed)
mrcnn_mask              (TimeDistributed)
```

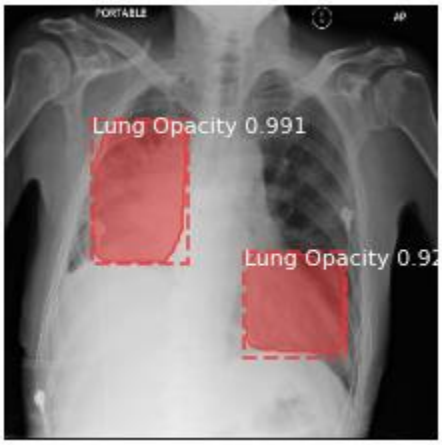
Metrics:

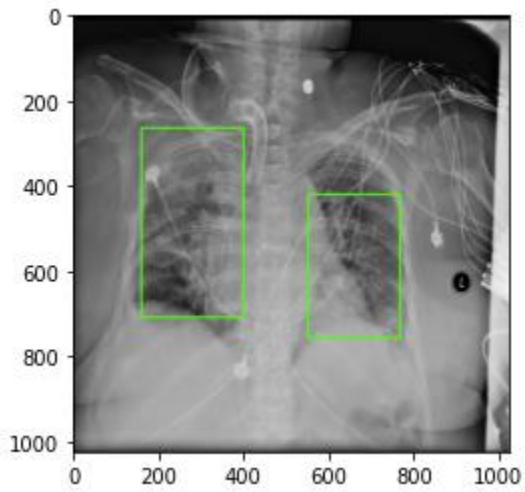
	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss
	loss	rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
1	2.097349	0.024572	1.048341	0.248470	0.454224	0.372331
	1.855555	0.025953	0.582310	0.283494	0.553975	0.409821
2	2.016423	0.025372	0.845828	0.214349	0.470426	0.380325
	2.006595	0.029374	0.852154	0.241419	0.493802	0.389841
3	1.797806	0.022649	1.005153	0.153250	0.437079	0.374107
	1.652955	0.020897	0.554882	0.231787	0.448598	0.396780
4	1.865097	0.018412	0.469107	0.217477	0.431760	0.367876
	1.634317	0.019300	0.540045	0.241296	0.439788	0.393874
5	1.788138	0.016778	0.535514	0.273547	0.439282	0.378708
	1.548045	0.017552	0.483424	0.223170	0.432816	0.391068
6	1.475289	0.019091	0.460752	0.210928	0.464885	0.387532
	1.549663	0.016468	0.488943	0.220015	0.433200	0.391022
7	1.264199	0.014916	0.502126	0.155293	0.384783	0.371378
	1.338019	0.014679	0.373491	0.176226	0.392625	0.380982
8	1.565714	0.013113	0.417052	0.160723	0.399444	0.372268
	1.291166	0.013319	0.364257	0.167670	0.373758	0.372145
9	1.441193	0.013003	0.451895	0.154401	0.388485	0.365647
	1.265051	0.011968	0.353448	0.152605	0.373444	0.373570
10	1.353702	0.013312	0.477395	0.165840	0.376577	0.364003
	1.259397	0.011316	0.349174	0.161515	0.364636	0.372739
11	1.433473	0.011156	0.359601	0.176389	0.359806	0.362395
	1.356441	0.013260	0.428271	0.176871	0.364892	0.373130
12	1.514940	0.012917	0.383283	0.183516	0.378218	0.364682
	1.309800	0.011965	0.397450	0.161582	0.369402	0.369384
13	1.054174	0.011842	0.419036	0.129460	0.372077	0.359025
	1.297916	0.011594	0.374509	0.165898	0.370288	0.375611
14	1.155423	0.012894	0.444421	0.130010	0.365587	0.361199
	1.269543	0.011773	0.374182	0.154944	0.358754	0.369873
15	1.206312	0.011280	0.365467	0.161572	0.366200	0.365953
	1.218855	0.011267	0.347396	0.140997	0.353639	0.365540
16	1.372300	0.011544	0.422079	0.161561	0.363465	0.354859
	1.247517	0.010631	0.355903	0.161007	0.355743	0.364215
17	2.265813	0.011875	0.435763	0.154921	0.386610	0.366657
	1.271162	0.010733	0.359224	0.162835	0.368660	0.369693
18	1.183721	0.011604	0.357709	0.179312	0.369631	0.355209
	1.244787	0.009698	0.346688	0.162230	0.357541	0.368613
19	1.218999	0.010267	0.361714	0.142381	0.365587	0.366033
	1.271570	0.010948	0.366702	0.169318	0.358790	0.365795
20	1.446321	0.010318	0.358432	0.129938	0.360530	0.359866
	1.209856	0.010456	0.331667	0.152432	0.352069	0.363215

Model Execution Time:

Compared all other models Resnet50 using Mask_RCNN approach model training very fast (due to using GPU compatible parameters) and provided the better results even the overall accuracy given as 88%.

Predictions:





Overall UNET model and customised Basic CNN with LeakyRelu approach given best accuracy more than 97% and Resnet Model with MASK_RCNN given best performance with best predictions with little less accuracy compared to UNET model.

Models and the Accuracy comparison:

Model	**Image Resolution**	**Accuracy**
Basic CNN with LeakyRelu	256x256	0.9751
UNET	224x224	0.9728
Basic CNN with Relu	224x224	0.9271
Resnet50 (MASK_RCNN)	256x256	0.88
VGG16	384x384	0.84
InceptionV3	224x224	0.68

6. Possible Improvements and recommendations

Documenting possible improvements for future attempts of this problem. This was done due to time restrictions of project timelines.

There are several ways we could have improved performance. To name a few:

- Using different augmentation parameters
- Train for more epochs (due to limited computational power not able train more epochs now)
- Optimizing the hypermeters using Grid/Random Search approach
- Using other pretrained model Like AlexNet, Densnet, MobileNet using pretrained weights etc
- Image segmentation models like Tiny_Yolo, Yolo3 and Yolo4, Detectron2 etc which are recent segmentation approaches which gives faster and very good results
- Sampling the data to get the equal amount of dataset

In order to try out so many ideas, we would need better hardware. This project was implemented using a Google colab and Kaggle GPUs, which was provided by the Colaboratory environment. Another alternative to our method, would be using pre-built tools that provide state-of-the-art object detection algorithms, such as TensorFlow Object Detection API or Detectron.

7. Summary

This project gave me a lot of new concepts to digest and a lot of experimenting with code. Overall, it was a really good experience and helped us improve fast over a short period of time. Adequate results were achieved through experimenting on different models and machine learning techniques. Most noticeably, hardware and software limitations. To sum it up, it would be fair to say that deep learning technologies have opened paths to attempt to solve problems in areas such as healthcare and medicine. This was a challenging and exciting problem and I will certainly experiment more and try new ideas in order to get better performance. As part of next I will use most recent segmentation model and the techniques to improve overall accuracy and performance.

With automation at the level of experts, we hope that Machine Learning and Artificial Intelligence can improve healthcare delivery and increase access to medical imaging expertise in parts of the world where access to skilled radiologists are limited.