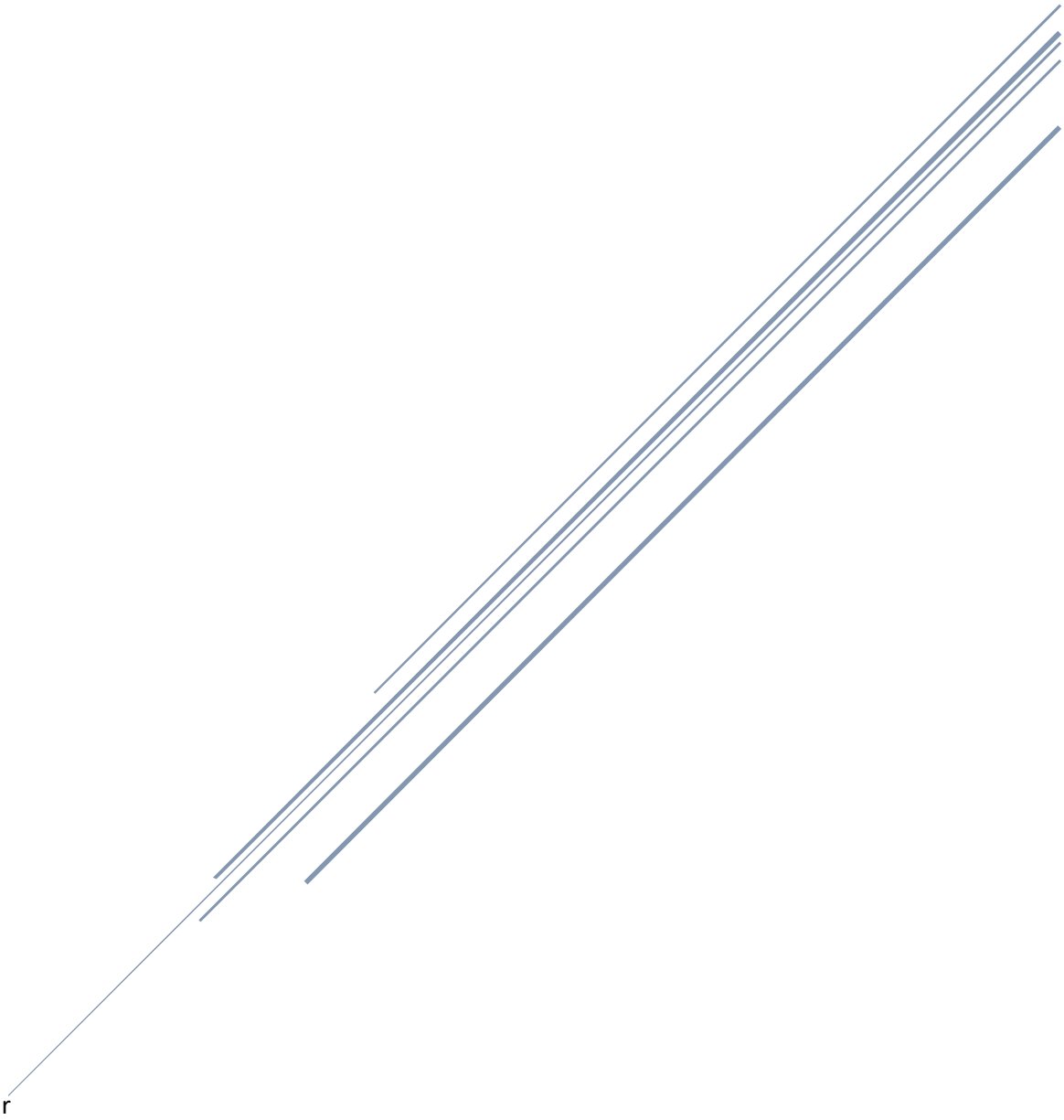
PNEUMONIA DETECTION

Analysis of Various Model for Prediction and Segmentation



Team

Kabardhi

Vani

Ashwathi

Sekar

Nandhakumar

Table of Contents

Problem Statement:

Pneumonia is an infection in one or both lungs. Bacteria, viruses, and fungi cause it. The infection causes inflammation in the air sacs in your lungs, which are called alveoli. 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 labeled “Not Normal No Lung Opacity”. This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia.

Data Format and attributes in the data:

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.

The below screen shot obtained during data exploration gives us the meta data present in files:

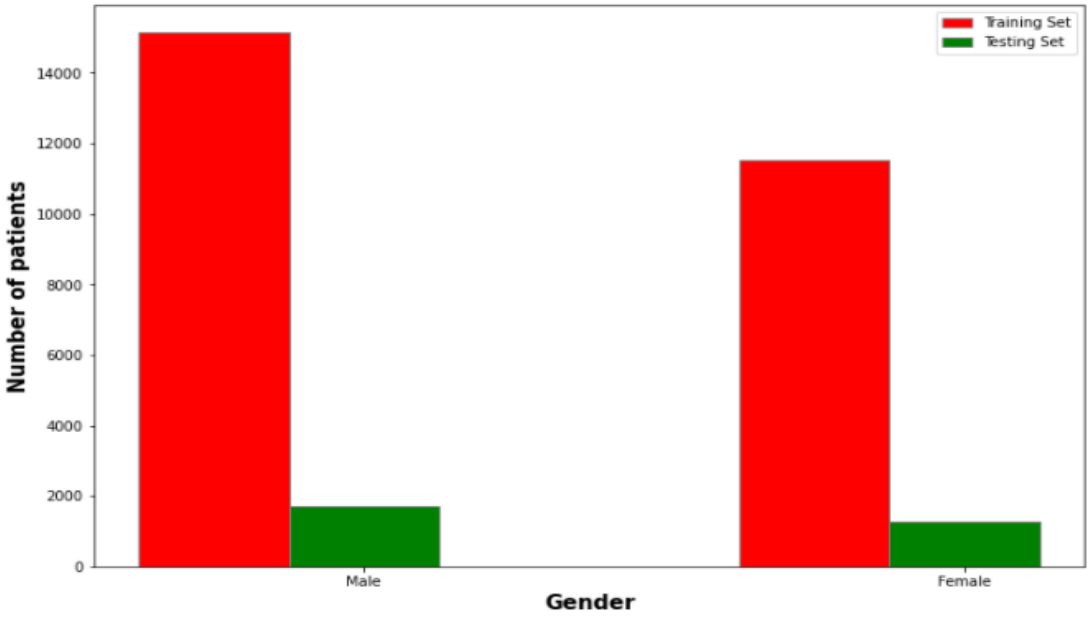
```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 202
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.23283.1517874447.220653
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.23283.1517874447.220653
(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: 'e77a7c87-8989-4dcc-8dc4-2e7cc98a5cc5'
(0010, 0020) Patient ID                          LO: 'e77a7c87-8989-4dcc-8dc4-2e7cc98a5cc5'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '71'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.23283.1517874447.220652
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.23283.1517874447.220651
(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 156288 elements
```

Out of the meta data present in the files, the ones that are of most value are:

- Patient Id: The DICOM files are named according to the patient-id
- Sex of the patient under examination
- Age of the patient under examination
- Pixel Data: Actual X-Ray of the patient
- Size of X-ray: 1024x1024 for all patients

Data Statistics:

1. The number of DICOM files provided in the training set is 26684
2. The number of DICOM files provided in the testing set is 3000
3. Data split according to the gender is equal in both the training and testing set. The following plots illustrate how the data is distributed according to the genders



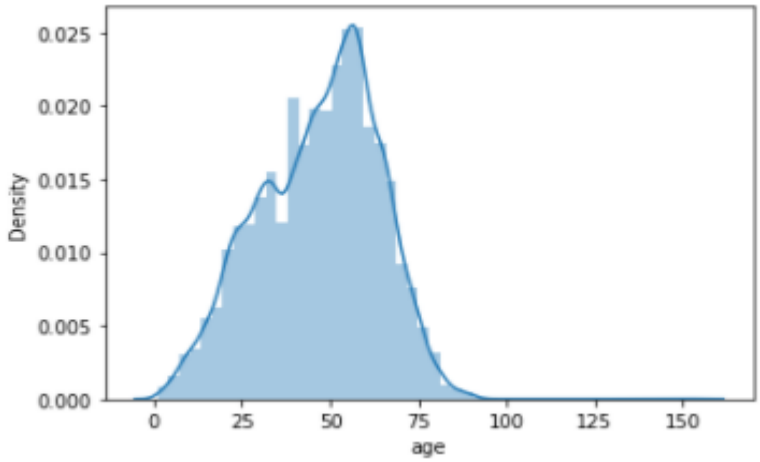
	Training Set	Testing Set
Number of Male Patients	15166	1714
Number of Female Patients	11518	1286
Total	26684	3000

Percentage of Males data in training set:56.8%
Percentage of Females data in training set:43.2%

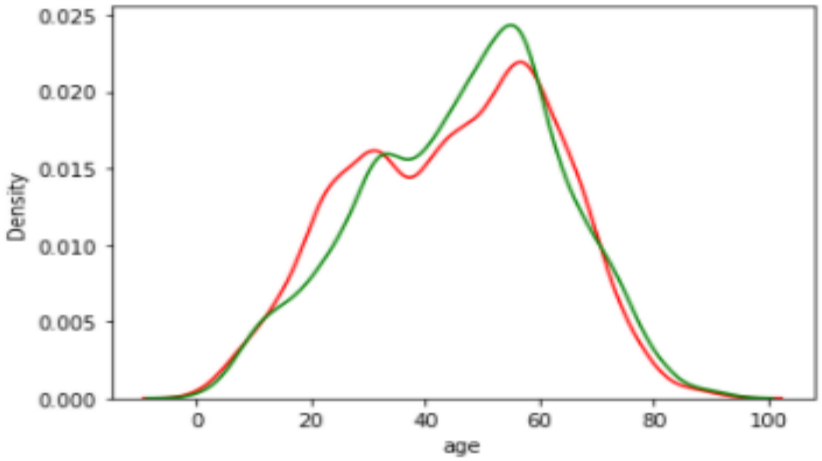
Percentage of Males data in testing set:57.1%
Percentage of Females data in testing set:42.9%

From the above numbers we can see that there is a fair distribution of data according to the gender of the patients.

4. When we look at the distribution of data in terms of age of the patients the distribution has peak around 55yrs indicating that the age group near to 55yrs could be more vulnerable to having pneumonia and have been tested more. The distribution is plotted below:



The above-mentioned point is valid for both the genders. It is seen that both male and female patients are at risk of pneumonia if their age is around 55-60years.Below plot has only data from patients with pneumonia.

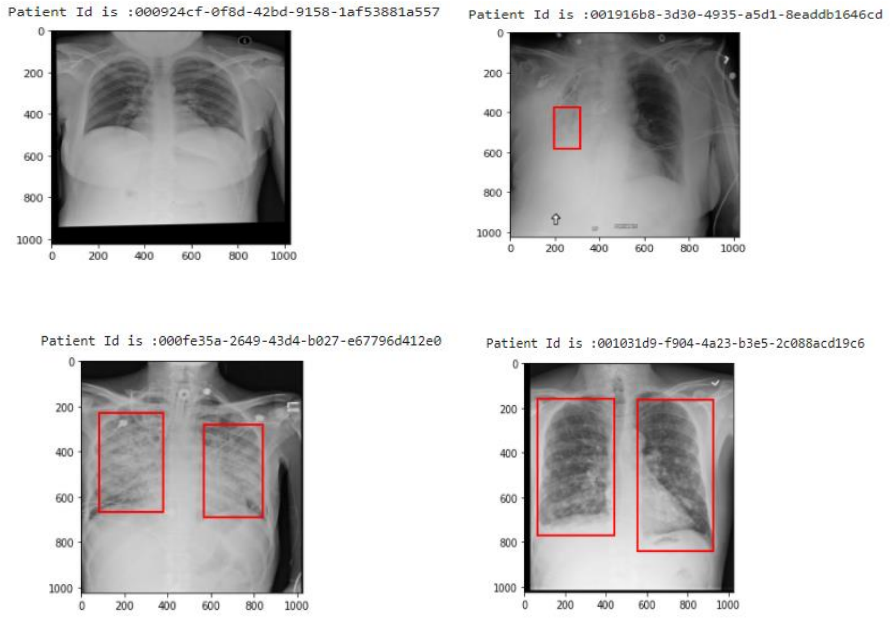


Labels:

The csv file provided with the labels contained the bounding box information and another label was provided describing the class. They both were merged into one data frame as shown below:

patientId	x	y	width	height	Target	class
0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
000924cf-0f8d-42bd-9158-1af53881a557	NaN	NaN	NaN	NaN	0	Normal
000db696-cf54-4385-b10b-6b16fbb3f985	316.0	318.0	170.0	478.0	1	Lung Opacity
000db696-cf54-4385-b10b-6b16fbb3f985	660.0	375.0	146.0	402.0	1	Lung Opacity
000fe35a-2649-43d4-b027-e67796d412e0	570.0	282.0	269.0	409.0	1	Lung Opacity
000fe35a-2649-43d4-b027-e67796d412e0	83.0	227.0	296.0	438.0	1	Lung Opacity
001031d9-f904-4a23-b3e5-2c088acd19c6	66.0	160.0	373.0	608.0	1	Lung Opacity
001031d9-f904-4a23-b3e5-2c088acd19c6	552.0	164.0	376.0	676.0	1	Lung Opacity
0010f549-b242-4e94-87a8-57d79de215fc	NaN	NaN	NaN	NaN	0	Normal
001916b8-3d30-4935-a5d1-8eaddb1646cd	198.0	375.0	114.0	206.0	1	Lung Opacity

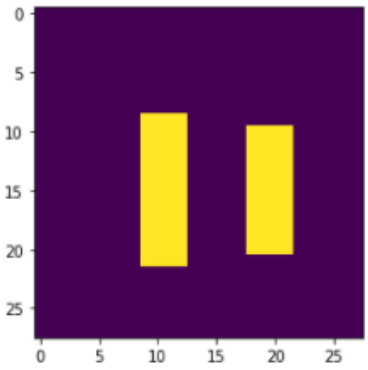
Also, one thing noticed is that though there were 26684 patient-ids in training set, but in the labels it is found that there were 30227 rows indicating either duplicate information. After inspecting we found that the there is no duplicate information but for a particular patient, we could have inflammation detected in both the lungs. Some examples for different cases are shown below:



One thing to notice here is that the bounding boxes come in different sizes.

Data Preparation for Model:

1. The images have been resized to have a size of 224x224. Since these are gray scale images, we have only one channel.
2. Using the bounding box information we have prepared the segmentation mask(28x28) which look like below (samples provided):



A sample segmentation mask.

MOBILENET

Architecture

Total params: 3,230,657
Trainable params: 1,793
Non-trainable params: 3,228,864

Weights used = ‘imagenet’

Hyper paramerts

Hyper-parameters for optimizer:

Optimizer used: Adam

- learning_rate=1e-3
- beta_1=0.9
- beta_2=0.999
- epsilon=1e-8
- decay=0.0
- amsgrad=False

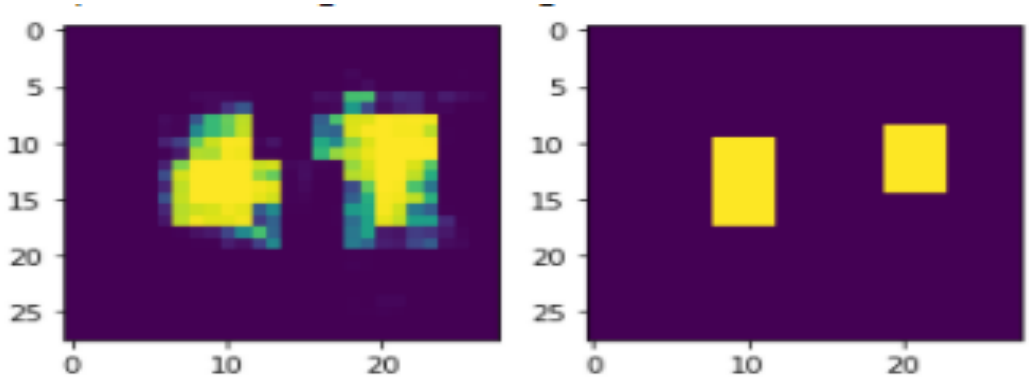
EarlyStopping used while monitoring "val_loss".
ReduceLROnPlateau also used.

Metrics

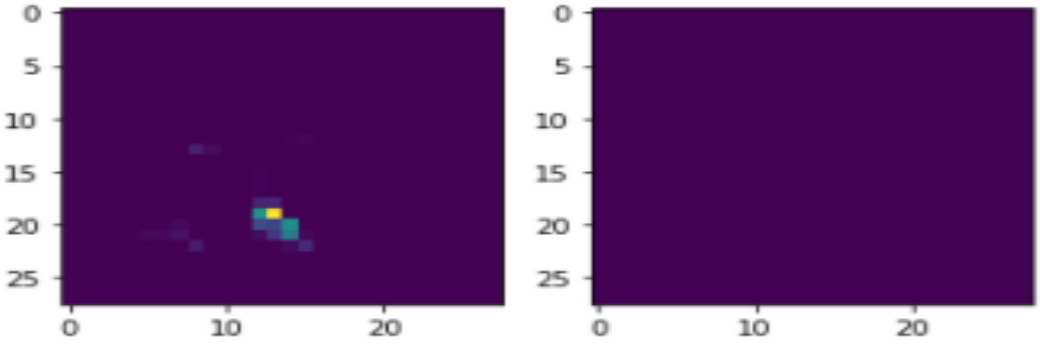
The dice coefficient we are getting on the model is 0.6

On the left we have the segmentation mask generated by the model and on the right we have the actual mask.

- Case 1: Target=1 that is, pneumonia detected



Case 2: Target=2 that is, pneumonia not detected



DENSENET

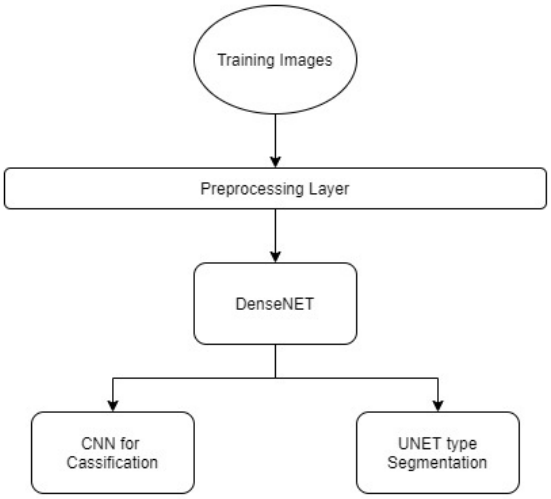
Preprocessing

All the training dicom files were preprocessed and extracted the image array to train the backbone model DenseNet.

Since the preprocessing was taking more CPU RAM in colab, we have moved the preprocessing step inside the model so that the preprocessing step will run on GPU to save memory in CPU. This provided better results and hence the colab was not crashing because of lesser memory.

Architecture

The Backbone model was densenet architecture with out the head/last layer output. The last layer was customized so that it can contain two heads – First to predict the classification problem, whereas the later will perform the segmentation problem.



Double Head Model

```
layer8x8 = model.get_layer('conv5_block16_2_conv').output
layer16x16 = model.get_layer('conv4_block24_2_conv').output
layer32x32 = model.get_layer('conv3_block12_2_conv').output
layer64x64 = model.get_layer('conv2_block6_2_conv').output
layer128x128 = model.get_layer('conv1/conv').output

output_segment = Concatenate()([UpSampling2D()(layer8x8), layer16x16])
output_segment = Concatenate()([UpSampling2D()(output_segment), layer32x32])
output_segment = Concatenate()([UpSampling2D()(output_segment), layer64x64])
output_segment = Concatenate()([UpSampling2D()(output_segment), layer128x128])
output_segment = UpSampling2D()(output_segment)
output_segment = Conv2D(16, kernel_size = 1,activation='relu')(output_segment)
output_segment = Dropout(0.2)(output_segment)
output_segment = Conv2D(7, kernel_size = 1,activation='relu')(output_segment)
output_segment = MaxPooling2D()(output_segment)
output_segment = MaxPooling2D()(output_segment)
output_segment = Conv2D(1, kernel_size = 1,activation='sigmoid', name='output_segment')(output_segment)



output_classify = Concatenate()([UpSampling2D()(layer8x8), layer16x16])
output_classify = Concatenate()([UpSampling2D()(output_classify), layer32x32])
output_classify = Concatenate()([UpSampling2D()(output_classify), layer64x64])
output_classify = Flatten()(output_classify)
output_classify = Dropout(0.2)(output_classify)
output_classify = Dense(2, activation='softmax', name='output_classify')(output_classify)

model = Model(model.input, [output_classify, output_segment])
```

Metrics

We have considered the Dice Coefficient and custom loss function for Segmentation problem and used Categorical cross entropy for classification problem.

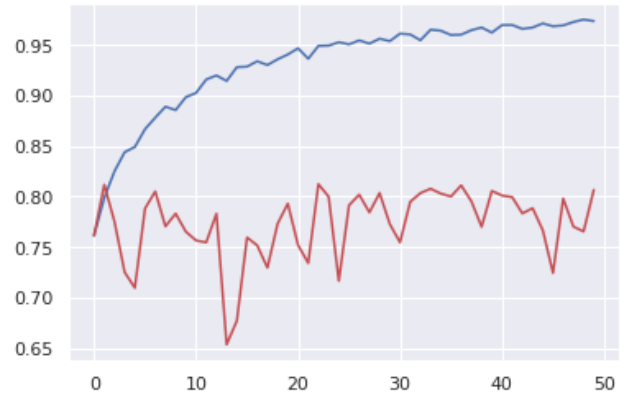
```
metrics = {
    "output_segment" : dice_coef,
    "output_classify" : "accuracy"
}

model.compile(loss = losses, optimizer='adam', metrics = metrics)

hist = model.fit(
    x = X_train,
    y = {"output_classify": y_train_classify, "output_segment": y_train_segment},
    epochs=50,
    batch_size=16,
    validation_data=(X_val, {"output_classify": y_val_classify, "output_segment" : y_val_segment }),
    verbose=1)
```

Performance

We have run across 20 epochs for this model, and were able to achieve better results. We tried with earlystops, with different back sizes sothat the training was apt for better results.

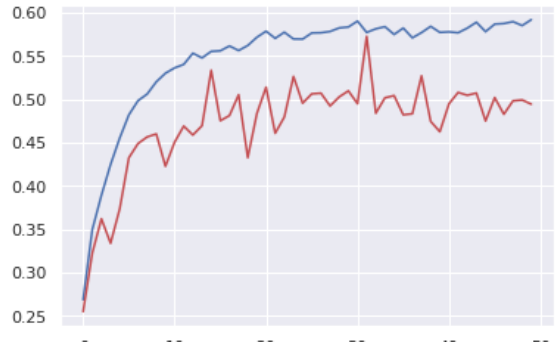
```
epochs = range(len(hist.history['output_segment_loss']))
plt.plot(epochs, hist.history['output_classify_accuracy'],'b', label = 'Accuracy')
plt.plot(epochs, hist.history['val_output_classify_accuracy'],'r', label = 'Accuracy')
```

```
[<matplotlib.lines.Line2D at 0x7fb8a629b190>]
```



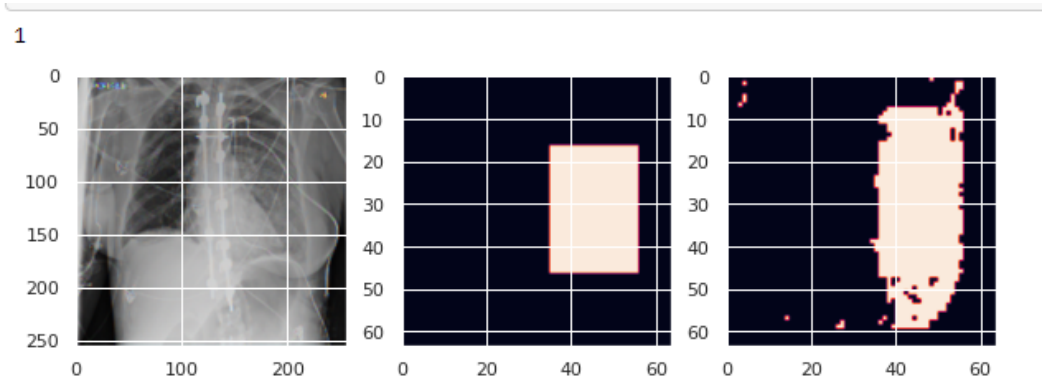

```
epochs = range(len(hist.history['output_segment_loss']))
plt.plot(epochs, hist.history['output_segment_dice_coef'],'b', label = 'Mean Squared Error')
plt.plot(epochs, hist.history['val_output_segment_dice_coef'],'r', label = 'Mean Squared Error')
```

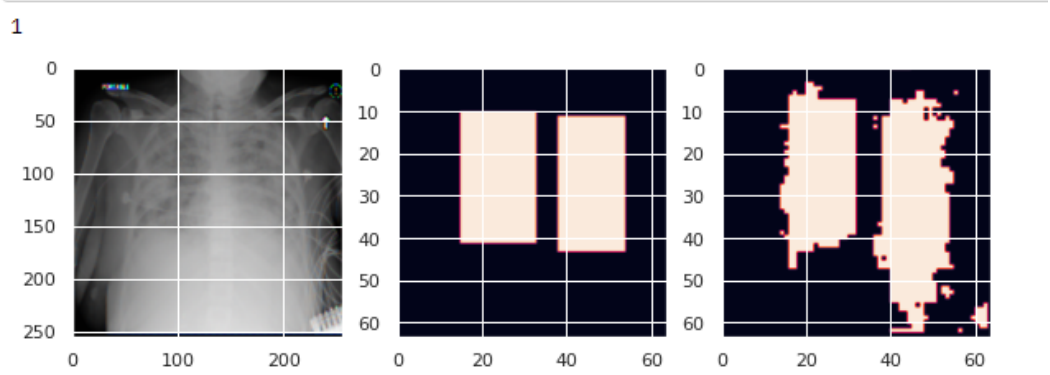
```
[<matplotlib.lines.Line2D at 0x7fb7a14f8750>]
```



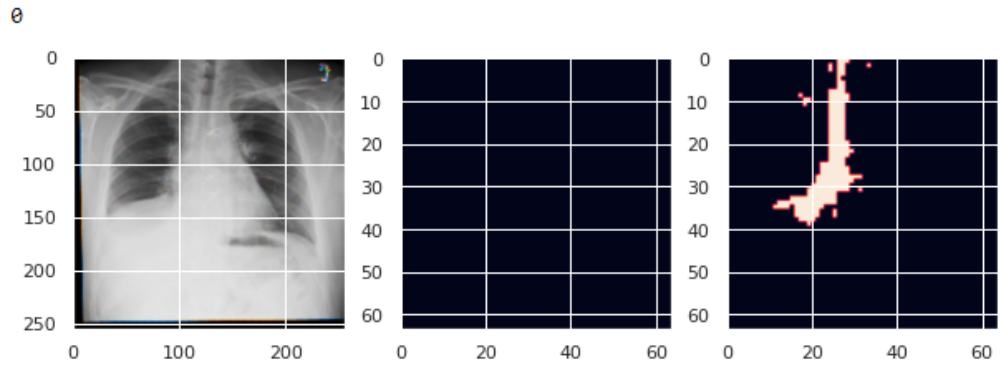
Sample Output

Detected Images





Non-Detected Images



YOLO

Object Detection using YoloV3

YOLO, short for You Only Look Once, is a popular State of the Art Model for Object Detection. The algorithm is a one stage approach that uses convolutional neural network to predict bounding boxes coordinates on image dataset. The outputs include class type and confidence score.

There are several versions of Yolo and the latest version is YoloV4 available on pjreddie and AlexeyAB github repositories. For this project, we have used the keras-yolov3 version from Huynh Ngoc Anh available at the following link:

experiencor/keras-yolo3: Training and Detecting Objects with YOLO3 (github.com)

This implementation is based on darknet architecture available from above official repositories. We have also referred the following blog article to get a basic understanding of the yolo model and its usage: How to Perform Object Detection With YOLOv3 in Keras (machinelearningmastery.com)

Installation

There are several steps in using the yolov3-model for pneumonia detection.

1) Clone the repository
2) Install the dependencies
3) Detect a test image
4) Custom object detection for pneumonia use case
    a. Pneumonia dataset preparation – images and annotations
    b. Edit the configuration file
    c. Generate anchors for the pneumonia dataset
    d. Train data on pneumonia dataset
    e. Perform detection using trained weights
5) Observations and next steps

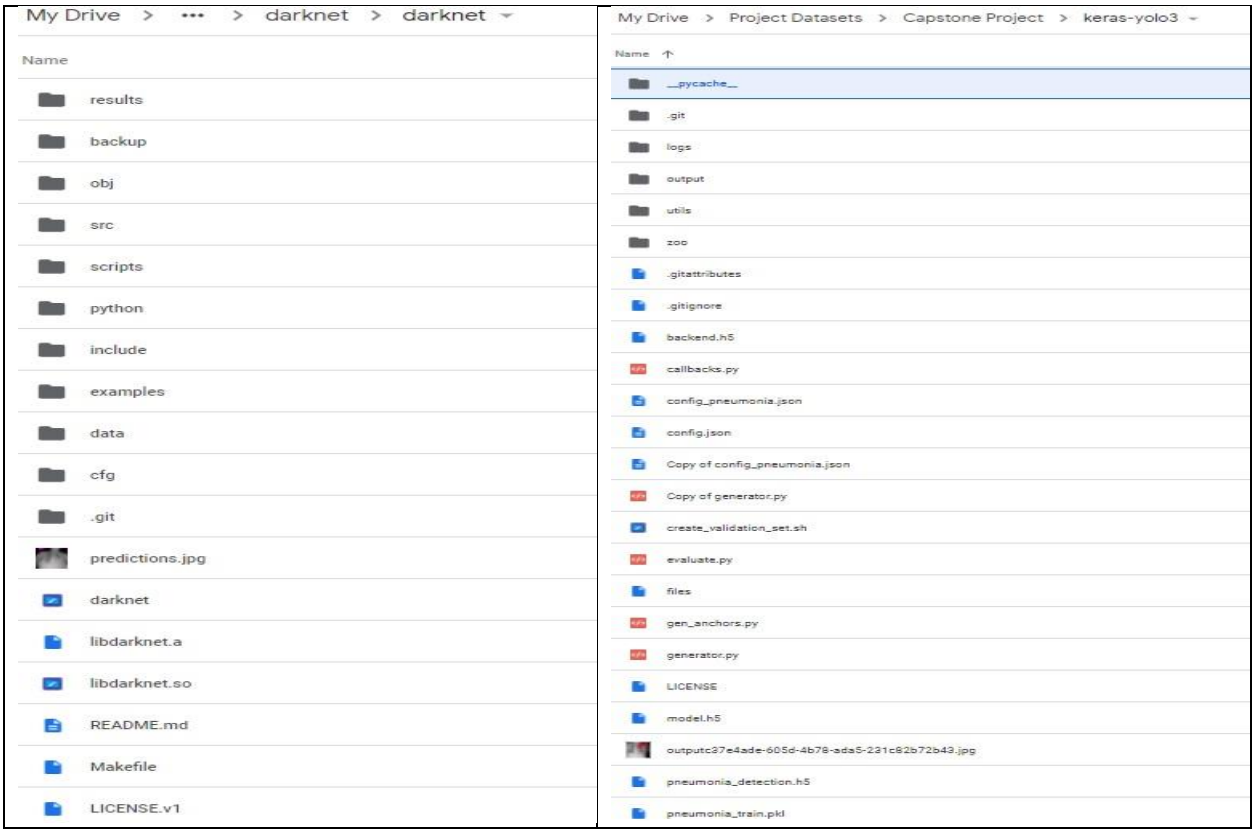
The above steps are detailed below:

Clone the repository

We cloned both the repositories to experiment with the functionality. However, we used the experiencor version for subsequent processing as the configuration process for custom object training is relatively simpler.

```
# !git clone https://github.com/pjreddie/darknet
# !git clone https://github.com/experiencor/keras-yolo3.git
```

The above commands create a folder structure such as shown below:



Install the dependencies

This is an important step as the yolov3 implementation code in python has references to functionality from these libraries. The tensorflow(1.15), keras(2.3.1), hdf5(2.10.0) and other libraries need to be at the required versions in order for the code to work.

```
!pip install -r requirements.txt
```

It would be to good to create

A virtual environment can be created to install the dependencies so that we can keep the ongoing development related to this program separate from other programs which may have other dependencies.

Detect a test image

Before training yolov3 network on the pneumonia custom dataset, it is important to check whether the installation and dependencies have completed properly. We can do this by predicting the result for a test image.

Before detecting a test image, we need to provide name scope by adding prefix tensorflow to all the import statements where keras libraries are mentioned. Once this is done, we test the yolov3 model on a test image named dog.jpg.

```
!python yolo3_one_file_to_detect_them_all.py -
w '/content/drive/MyDrive/Project Datasets/Capstone Project/darknet/yolov3
.weights' -
```

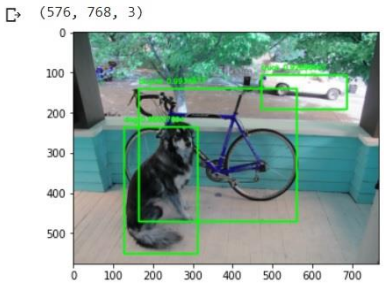
```
i '/content/drive/MyDrive/Project Datasets/Capstone Project/darknet/darkne
t/data/dog.jpg'
```

The above code uses the default yolov3 weights available from darknet repository and the dog image available as part of data directory.

This provides the following output:

```
bicycle: 99.34676885604858%
dog: 98.6376941204071%
truck: 92.89804697036743%
```

Then, we use matplotlib and cv2 to display the image and bounding boxes.



Custom object detection

Data Preparation

This step includes creation of separation folders for train and validation images and their corresponding annotations. If the validation folder is empty, then training set will be automatically splitted in the ratio of 0.8.

For this project, we used only a subset of images for training purpose considering memory requirements and speed of execution. Dicom images were preprocessed separately and converted to jpg images of size 256 x 256 and used as the input. The following code copies subset of such images into the training_images folder

```
# Code to copy a subset of images from source folder to train_images_pr
esent
import shutil, os
source_dir = "/content/drive/MyDrive/Project Datasets/Capstone Project/
jpg_reshaped_train/present/"
dest_dir = "/content/drive/MyDrive/Project Datasets/Capstone Project/tr
ain_images_present/"
files = os.listdir(source_dir)[:500]
for f in files:
    shutil.copy(source_dir + f, dest_dir)
```

Once images are available, the next step is to create annotations in VOC format. Pascal-voc-writer was used to create annotations in VOC format. Another key input to generate annotations includes the

bounding box coordinates for the train image set. This data was preprocessed separately and kept ready in a pickled file which was loaded and stored in a dataframe. A brief stub was written to read the dataframe and convert the bounding box inputs into VOC format. This included consideration for images with multiple bounding boxes in a single patient image.

Example of one such annotation file generated is shown below:

```
<annotation>
    <folder>train_images_present</folder>
    <filename>c37cfadd-1e52-4704-9831-403826ad2974.jpg</filename>
    <path>/content/drive/MyDrive/Project                  Datasets/Capstone
Project/train_images_present/c37cfadd-1e52-4704-9831-
403826ad2974.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>256</width>
        <height>256</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>opacity</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>60.0</xmin>
            <ymin>39.0</ymin>
            <xmax>129.0</xmax>
            <ymax>174.0</ymax>
        </bndbox>
    </object>    <object>
        <name>opacity</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>167.0</xmin>
            <ymin>40.0</ymin>
            <xmax>237.0</xmax>
            <ymax>157.0</ymax>
        </bndbox>
    </object>
</annotation>
```

Edit the configuration file

This is one of the most important steps as the configuration file is the key for generating image anchors and for training process.

```
{
    "model" : {
        "min_input_size":       288,
        "max_input_size":       448,
        "anchors":              [56,
52,  65,97,  71,146,  85,68,  94,194,
98,138, 107,102, 119,168, 122,221],
        "labels":
["opacity"]
    },

    "train": {
        "train_image_folder":
"/content/drive/MyDrive/Project
Datasets/Capstone
Project/train_images_present/",
        "train_annot_folder":
"/content/drive/MyDrive/Project
Datasets/Capstone
Project/train_annotations_present/",
        "cache_name":
"pneumonia_train.pkl",
        "pretrained_weights":
"backend.h5",
        "train_times":          8,
        "batch_size":           8,
        "learning_rate":         1e-
4,
        "nb_epochs":            10,
        "warmup_epochs":        3,
        "ignore_thresh":        0.5,
        "gpus":                 "0",

        "grid_scales":
[1,1,1],
        "obj_scale":            5,
        "noobj_scale":          1,
        "xywh_scale":           1,
        "class_scale":          1,
        "tensorboard_dir":
"logs",
        "saved_weights_name":
"pneumonia_detection.h5",
```

Model section has the following parameters:

- Min_input_size and Max_input_size are used to define the range within which the images are resized
- The anchor box sizes are used to improve the quality of object detection. The coordinates of anchor boxes mentioned in the configuration file are obtained after running the command described in 4c below.
- Labels tag is updated with one class name 'opacity' corresponding to pneumonia dataset.

Train section has the following parameters:

- Train_image_folder – location of images to be trained with full path
- Train_annot_folder – location of image annotations which correspond 1:1 with the images itself
- Cache_name – the name of the file where annotation results are stored. This will get created first time we run this
- Pretrained_weights – we need to use the backend weights available from darknet repository
- Train_times - the number of time to cycle through the training set. Retained the default value of 8
- Batch_size - # the number of images to read in each batch. This is a very important parameter to fine tune considering available memory
- Learning Rate - # the base learning rate of the default Adam rate scheduler
- Nb_epochs – number of epochs to train the model

<pre>"debug": true
},
 	
"valid": {
"valid_image_folder": "",
"valid_annot_folder": "",
"cache_name": "",
 	
"valid_times": 1
}
}</pre>
	• Gpus – flag to indicate to use gpu when the value is 0
	• Saved_weights_name – file to store the trained model with weights
	• Default values chosen for remaining parameters
	
	Since there is not separate validation dataset, we have not set these parameters

Training

Generate anchors for the pneumonia dataset:

Anchors can be generated using the following command:

```
!python gen_anchors.py -c config_pneumonia.json
```

```
The script gen_achors.py generated the following result:

Average IOU for 9 anchors: 0.83
56,52, 65,97, 71,146, 85,68, 94,194, 98,138, 107,102, 119,168, 122,221
```

Train data on pneumonia dataset

Training performed using the following command:

```
!python train.py -c config_pneumonia.json
```

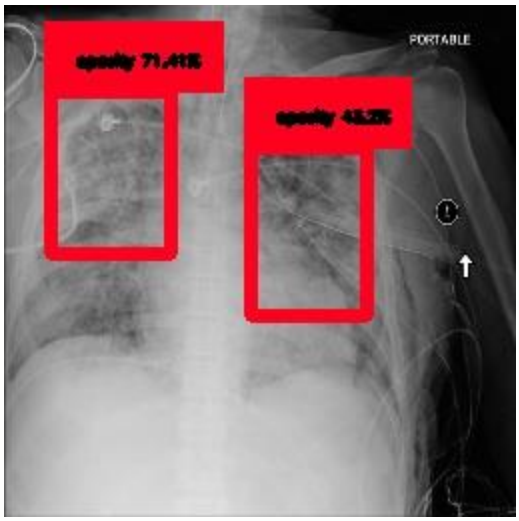
Once training is completed, the script prints out mAP value.

Prediction

Perform detection using trained weights:

Training perform on a test image using the following command saves the result image with predicted bounding boxes in the output folder(default).

```
!python predict.py -c config_pneumonia.json -
i '/content/drive/MyDrive/Project Datasets/Capstone Project/train_image
s_present/c37e4ade-605d-4b78-ada5-231c82b72b43.jpg'
```



Observations and next steps

The mAP value is very low considering the number of images trained. By training additional images and increasing number of epochs, adjusting the values of object threshold and nms threshold, we can increase prediction quality.

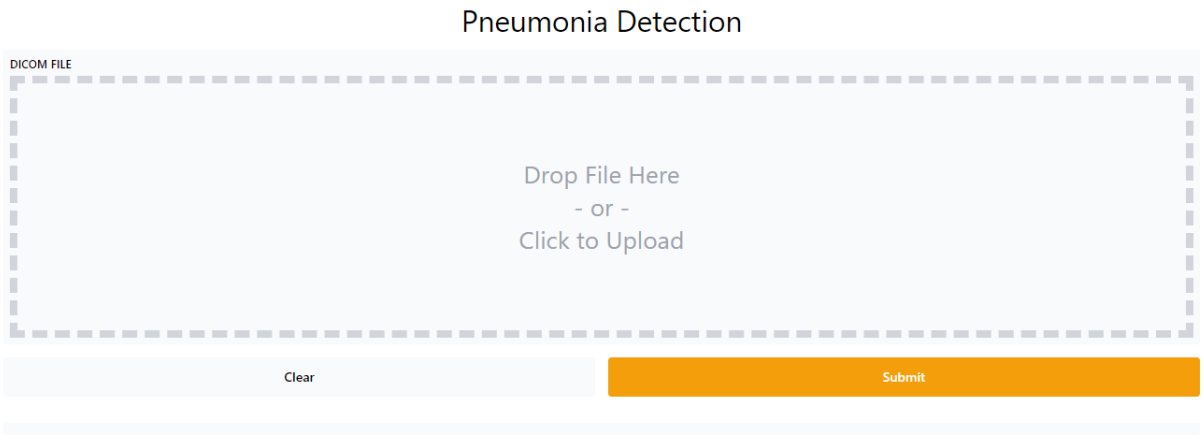
Model deployment & User Interface

User Interface

We have used Gradio for development of UI. This interface helps the Data scientists to create build the UI in a fast pace to test the models.

```
47
48  iface = gr.Interface(fn = detectPneumonia,
49                       inputs = [
50                               gr.inputs.File(label="DICOM File")
51                       ],
52                       outputs = [
53                               gr.outputs.KeyValues(label="Patient Details"),
54                               gr.outputs.Image(label="Loaded Image"),
55                               gr.outputs.Image(label="Segmented Image"),
56                               gr.outputs.Label(label="Is Detected"),
57                       ],
58                       layout = "vertical",
59                       title = "Pneumonia Detection",
60                       allow_flagging = False)
61  iface.launch()
```

User Interface Input



User Interface Output

Deployment

We have used DenseNet Model for UI Implementation. We have used Heroku as a deployment platform

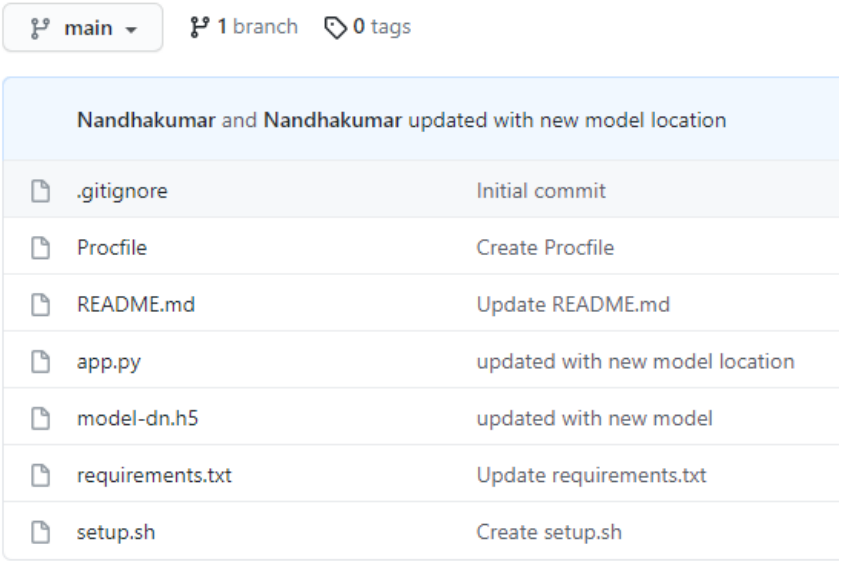
Below are the steps used to start the web app.

1. Create app.py with gradio interface
2. Create requirements.txt file
3. Create setup.sh with below config export GRADIO_SERVER_NAME=0.0.0.0 export GRADIO_SERVER_PORT="$PORT"
4. Create Procfile which initiates the whole process web: source setup.sh && python app.py
5. Now deploy to heroku app

Code Repository

Deployment URL

http://pne-lung-gradio.herokuapp.com/



UI App Repository

https://github.com/nandhukumar86/pne-lung-gradio

EDA and model Repository

https://github.com/nandhukumar86/CapstonePneumoniaDetection