FINAL REPORT

AIML CAPSTONE PROJECT

Computer Vision - PNEUMONIA DETECTION CHALLENGE

FINAL SUBMISSION BY

Gopi Krishna, Haritha, Ankur Saxena, Hari Kumar M, Priya, Shailesh Choudhari and Anand Vatkar

1. Summary of problem statement, data and findings

Problem Statement:

In the domain of Health Care, identifying diseases through imaging is an essential part of diagnostics. The specific challenge described here is the detection of Pneumonia, a condition characterized by the inflammation of the lungs. The goal is to Design and develop a DL based computer vision algorithm capable of identifying and locating lung opacities, which are visual signals for pneumonia, on chest radiographs.

Data Description:

The dataset provided for this challenge consists of medical images stored in DICOM files (*.dcm), a specialized format that combines header metadata with underlying raw image arrays for pixel data. Some of the images are labeled as “Not Normal No Lung Opacity”, a category indicating that while pneumonia is not present, there is an abnormality on the image that might mimic the appearance of pneumonia. The dataset likely includes varying examples of lung conditions, aiding in the training of the algorithm to detect the specific visual cues associated with pneumonia.

Findings and Implications:

In this study, we explored different machine learning models to enhance the accuracy of medical image analysis for pneumonia detection. We considered various architectures, including Convolutional Neural Networks (CNN), VGG16, and Mask R-CNN, and performed experiments using two different image formats: DICOM (with 14,000 images) and PNG (with all images).

The results showed the significant impact of varying decision thresholds on performance metrics, with lower thresholds favoring higher recall and higher thresholds balancing accuracy and precision at the cost of recall. CNN models generally exhibited shorter execution time than more complex models like VGG16 and Mask R-CNN.

Additionally, our investigation revealed that the conversion of images to PNG affected performance differently across models, and the threshold management played a crucial role in balancing recall with other parameters. The trade-offs between recall, accuracy, and precision underscored the need for careful threshold management, particularly in medical applications where high recall is vital to reduce false negatives.

Furthermore, the comparison of execution times among models emphasized the importance of considering this aspect for real-world applications, especially when dealing with large-scale datasets. The

study also identified areas for future improvement, including fine-tuning models, threshold selection, and tailored preprocessing techniques.

Overall, the comprehensive analysis conducted in this study provides essential insights and sets a strong foundation for future advancements in medical image analysis, specifically for pneumonia detection. The findings highlight the importance of carefully selecting and tuning model architectures and preprocessing methods to meet the specific requirements of the task at hand.

2. Overview of the final process

Brief on methodology used for solving the problem:

Steps involved are as follows

1. Analyze, detect deviations, correct, convert and collect all data and information in one place by creating a metadata dataframe which links each image file to all information related to it.
2. Run a base CNN model (accomplished in Milestone 1) and analyze the results for
    a. Selecting metrics suitable for the problem at hand. It was realized that the recall has to be used as a prominent metric during as the significant class imbalance exists in the data provided.
    b. Selection of models and tweaking of the model architecture. In our case we have created various CNN model with different architectures. To improve classification performance, transfer learning VGG16 model was selected. For Object detection two models were selected. One based on resnet and Mask RCNN which predicts masks. And another one for classification, mask and bounding box prediction was selected. This model is based on Mask RCNN implementation based on matterport (Ref: https://github.com/leekunhee/Mask_RCNN)
3. Run the selected model and analyze the results. Special functions were developed for Analysing the results. Functions which printed, classification report, confusion matrix, Precision recall curve, ROC-AUC curve for each model.
4. Hyper parameter tuning- To improve the accuracy and recall, learning rate, model parameters like filter numbers, layers were varied. Further function was developed to study the impact of threshold on performance parameters

Data: The source of input data is 1. Data files provided and 2. Data from the DICOM image headers. Let's briefly discuss the features of both one-by-one

1. Data Files:

2 zip files containing DICOM image files for training and testing and three CSV files related to medical images are provided:

- stage_2_train_images.zip: This ZIP file contains all the DICOM files for the training set. we extract the ZIP file into a directory with the same name to allow the code to access these files.
- stage_2_test_images.zip: Similarly, this ZIP file contains all the DICOM files for the testing set.
- stage_2_train_labels.csv: Contains training labels.
- stage_2_detailed_class_info.csv: Contains detailed class information.

- stage_2_sample_submission.csv: Contains sample submission for test labels.

Data Description:

Images:

Training Images: A total of 26,684 DICOM images are used for training.

Test Images: A total of 3,000 DICOM images are used for testing.

CSV Files:

train_labels.csv:

Contains 30,227 records.

Includes bounding box coordinates (if present) and a binary target indicating the presence of lung opacity.

Some images have multiple bounding boxes, explaining why the number of records is greater than the number of unique images.

class_info.csv:

Contains 30,227 records.

Maps each image to one of three classes: 'No Lung Opacity / Not Normal', 'Normal', or 'Lung Opacity'.

As with train_labels.csv, multiple records can exist for a single image if multiple bounding boxes are present.

stage_2_sample_submission.csv:

Contains information about test images.

Specifies the required format for output submission.

Observations:

The number of records in both train_labels and class_info being 30,227, greater than the number of unique images, indicates that there are multiple instances where lung opacity is detected in a single image.

The number of unique images in both train_labels and class_info matches the number of training images, confirming consistency.

2. Data from DICOM Image headers:

DICOM images have important information about the patient, equipment and image itself stored in its header. It is essential to utilize the information properly to obtain the better results. .ipynb file goes deep in the details of the parameters and their distribution and impact. Here it sufficient to state that following parametsrs were studied Modality, BodyPartExamined, ViewPosition, ConversionType, PixelSpacing, PhotometricInterpretation, SamplesperPixel,

LossyImageCompression, LossyImageCompressionMethod, PixelRepresentation, Image Size, Patient's Sex, Age. The information and the research on it was useful in establishing,

1. All the images have same size, Modality, BopdyPartExamined, ConversionType, SamplesperPixel, LossyImageCompression, LossyImageCompressionMethod, PixelRepresentation
2. The images have been preprocessed in Compressed using lossy jpeg format. Hence images can be converted to other formats for use in training.
3. Since Photometric Interpretation is MONOCHROME2, minimum pixel value is intended to shown as black (0). This has to be taken care while normalizing the image.
4. there are 7 different PixelSpacing values in the dataset. But the research indicated that, CNN are scale invariant can handle the different pixel spacing. Hence, No processing was done to equalize PixelSpacing
5. ViewPosition was studied in detail and the distribution of bounding boxes was found to be somewhat different for each. For increasing the accuracy of the model further, this information can be used in the future. However our research indicated that the model are able to predict ViewPosition from images with 98% accuracy, implying that the CNN models learn to recognize the difference during training. Hence this information was not utilized in present study.
6. Impact of patient's Sex and Age diagnosis was studied and noted. The information can be utilized in the future for model performance improvement. Present study ignores this information.

Data Preprocessing:

Data preprocessing involves two aspects.

1. Creation of a metadata DataFrame to collate all above information in a single place. All relevant information associated with each unique image is kept in one row. This will facilitate further processing, modeling, and analysis. Dictionaries were be created to map information such as target, class label, and bounding boxes to the train images.
2. Since the data size is huge, 3GB+ and during initial run it was observed that the kernels were freezing for the lack of memory resources across all platforms, it was decided to convert the images to png format and store them in a folder structure suitable for flow_from_directyory method of imageGenerator function. This way demand on memory requirement was over come. And training on all possible could be done using all images. During preprocessing of the images one can chose to set the target size of the image (224, 512, 112) and also the color depth greyscale or RGB.

Algorithms used:

List of Models trained is given below. Please note that models 1 to 5 are trained on subset of DICOM images. 6-7 on full set of resized and converted PNG files. Model 8 is trained on DICOM images resized to size 256x256

A) Using a Limited Number of DICOM Images in Memory (14,000 images used):

1 CNN for classification - with Recall

2 CNN for classification - with Recall & Accuracy

3 VGG16 for classification with Recall & Accuracy

4 VGG16 for classification with Recall

5 Mask R-CNN, ResNet-based for mask prediction

B) Using All Images Converted to PNG Format (All images used):

6 VGG16 for classification - with Recall & Accuracy

7 CNN for classification - with Recall

C) Using All DICOM Images resized to 256x256

8 Mask R-CNN for classification, mask, and bounding box prediction

The code is designed to be modular. For each model only the model definition varies, rest of the code from compiling to data augmentation, training, plotting and displaying results is common. This way custom CNN models, transfer learning models, even mask RCNN model could be run from start to finish in one go[$] . Also the best models from each run was saved, which can be loaded separately, saving time for rerunning the already well trained model again.

$ The cell numbers are different as different models required significant amount of tuning and that section of the code was run many times. Modularity in code helped achieve this functionality.

3. Step-by-step walk through the solution

Describe the steps you took to solve the problem. What did you find at each stage, and how did it inform the next steps? Build up to the final solution.

Approach / Implementation

Data Reading and Pre-processing

Data pre-processing is the process of transforming raw data into an understandable format. Sometimes, the collected data dataset contains several different values, which lead to improper learning.

1. The data was presented in zipfile with train and test data. In order to process the data, unzipping of data is required as

```
# importing the zipfile module
from zipfile import ZipFile
def unzipfile(zipf):
    with ZipFile(zipf, 'r') as zObject:
        # Get the list of files names in the zip
        list_of_file_names = zObject.namelist()

        # Iterate over the list of file names in this case 'list_of_file_names'
        for file_name in tqdm(list_of_file_names, desc="Extracting "+zipf):
            # Extract each file in progress_bar
            zObject.extract(member=file_name)
```

Alongwith zipfiles, the labels (both train and test) and class info is provided. Next, is to assign correct image_id with train or test label.

```
# load labels from input files
train_labels = pd.read_csv('stage_2_train_labels.csv')
test_labels = pd.read_csv('stage_2_sample_submission.csv')
class_info = pd.read_csv('stage_2_detailed_class_info.csv')

# Define constants
ORIG_SIZE = 1024 # DICOM image original size
image_size = 224  # Size of input images
batch_size = 16 # set batch_size parameter
EPOCHS_SET = 12 # number of epochs to run

# Define paths for image io as per zip file
train_images_path = 'stage_2_train_images'
test_images_path = 'stage_2_test_images'

set_color = 'RGB' #'L'
n_classes = 2
id_str = str(image_size)+str(set_color)+str(n_classes)
# Define paths for image io for converted PNG files to make them suitable for flow from
train_png_images_path = 'train_images'+id_str
val_png_images_path = 'val_images'+id_str
test_png_images_path = 'test_images'+id_str

lbl2 = ['No_Opacity', 'Opacity']
lbl3 = ['Normal', 'No_Opacity_Not_Normal', 'Opacity']

# unzip dicom images

# Check if the directory exists before unzipping
if not os.path.exists(train_images_path):
    unzipfile(train_images_path+'.zip')

# Check if the directory exists before unzipping
if not os.path.exists(test_images_path):
    unzipfile(test_images_path+'.zip')
```

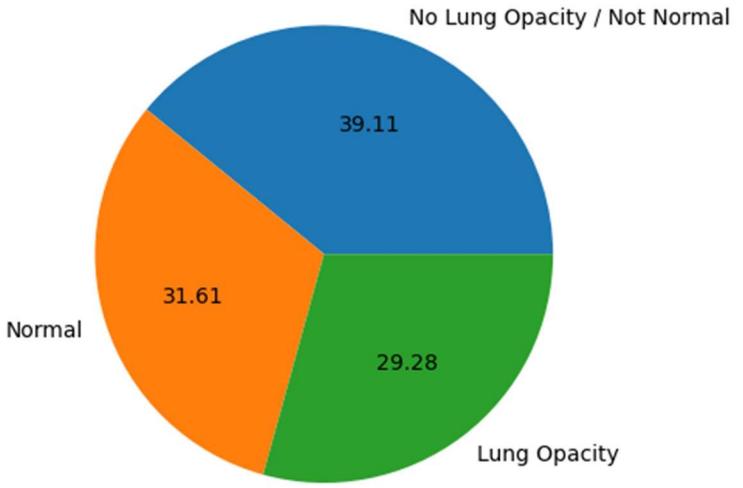
We observed that the class_info is provided with imbalanced datasets. This has been observed as

```
# get the distribution of targets to study if class imbalance exists

class_info['class'].value_counts()
```

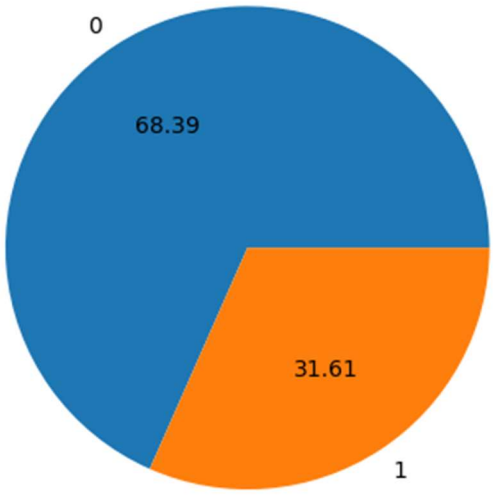
```
No Lung Opacity / Not Normal    11821
Lung Opacity                     9555
Normal                           8851
Name: class, dtype: int64
```

This imbalance is suggesting to have ambiguous information among lung opacity and normal classification. It looks like there is third set of X-ray which leading to multi-class dataset.

To visualize in a better way, it is evident from below image where this mult-classification is exists.



2. To understand the distribution of target within dataset. Here too we are observing imbalance and it is reflecting as below. And target value as 1 means chest xray is classified as with Pneumonia and target value as 0 means it is normal chest x-ray. When there is 'Lung Opacity' is labeled then the bounding rectangle coordinates are specified. Due to existence of same patientID, multiple regions are detected in a single image in both class_info and train_labels dataframe.



3. There is a genuine need to understand the shape of data. It is observed there are multiple bounding boxes in some images. The number of images and data labels implies the same. Total

images (both train and test) are 30227 and unique images are 29684 (train images - 26684 and test images - 3000). In order to map target, class label and bounding boxes in class_info and train_labels to train_images we have created dictionaries. Additionally, we are creating metadata dataframe which creates single place where all information is collated.

```
# Study the shapes
print(f'Train labels = {train_labels.shape}, Test labels = {test_labels.shape}, classes
```

```
Train labels = (30227, 6), Test labels = (3000, 2), classes = (30227, 2)
```

```
nTranRecs = train_labels['patientId'].nunique()
nInfoRecs = class_info['patientId'].nunique()
print(f'Unique images in train_labels = {nTranRecs} and in class_info = {nInfoRecs}')
```

```
Unique images in train_labels = 26684 and in class_info = 26684
```

4. Map train and test images to its classes. And map train images to its annotations.
   This has been carried out with the help below code as

```
# Step 2: Map training images to its classes

# Create a dictionary to map patientId to its class (Target)
train_class_dict = train_labels.groupby('patientId')['Target'].apply(list).to_dict()

# Create a dictionary to map patientId to its class info
train_class_info_dict = class_info.groupby('patientId')['class'].apply(list).to_dict()
```

```
import itertools
def first5of(dic):
    # Print the first 5 items
    first_five_items = pd.DataFrame(itertools.islice(dic.items(), 5))
    display(first_five_items)
```

```
first5of(train_class_dict)
```

	0	1
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	[0]
1	000924cf-0f8d-42bd-9158-1af53881a557	[0]
2	000db696-cf54-4385-b10b-6b16fbb3f985	[1, 1]
3	000fe35a-2649-43d4-b027-e67796d412e0	[1, 1]
4	001031d9-f904-4a23-b3e5-2c088acd19c6	[1, 1]

```
first5of(train_class_info_dict)
```

	0	1
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	[No Lung Opacity / Not Normal]
1	000924cf-0f8d-42bd-9158-1af53881a557	[Normal]
2	000db696-cf54-4385-b10b-6b16fbb3f985	[Lung Opacity, Lung Opacity]
3	000fe35a-2649-43d4-b027-e67796d412e0	[Lung Opacity, Lung Opacity]
4	001031d9-f904-4a23-b3e5-2c088acd19c6	[Lung Opacity, Lung Opacity]

5. Additionally, mapping of training and testing images to its annotations. This is to bring close to reality of training images. This allows machine learning model to learn, understand patterns and features in the data. Annotations help evaluate the performance of the model during testing. By comparing the model's predictions to the annotated ground truth, we can measure various performance metrics such as accuracy, precision, and recall. This evaluation is essential in determining the model's effectiveness and fine-tuning the algorithm

```
# Step 3: Map training images to its annotations

# Create a dictionary to map patientId to its bounding boxes (x, y, width, height)
# We first create a DataFrame where each row contains a list [x, y, width, height]
bounding_boxes = train_labels.dropna()\
                             .groupby('patientId')[['x', 'y', 'width', 'height']]\
                             .apply(lambda x: x.values.tolist()).reset_index()
# Rename the column
bounding_boxes = bounding_boxes.rename(columns={0: "bboxes"})

# The column name after rename operation is "bboxes", so we need to refer to that instea
train_anno_dict = dict(zip(bounding_boxes['patientId'], bounding_boxes["bboxes"]))
```

```
first5of(train_anno_dict)
```

	0	1
0	000db696-cf54-4385-b10b-6b16fbb3f985	[[316.0, 318.0, 170.0, 478.0], [660.0, 375.0, ...
1	000fe35a-2649-43d4-b027-e67796d412e0	[[570.0, 282.0, 269.0, 409.0], [83.0, 227.0, 2...
2	001031d9-f904-4a23-b3e5-2c088acd19c6	[[66.0, 160.0, 373.0, 608.0], [552.0, 164.0, 3...
3	001916b8-3d30-4935-a5d1-8eaddb1646cd	[[198.0, 375.0, 114.0, 206.0]]
4	0022073f-cec8-42ec-ab5f-bc2314649235	[[575.0, 232.0, 246.0, 528.0], [161.0, 230.0, ...

6. To perform preprocessing and infer through visualization, below mentioned steps are followed. Before proceeding with the next steps, it is better to understand more about DICOM images…

   DICOM (Digital Imaging and Communications in Medicine) is a standard used for storing and transmitting medical images. The DICOM standard was developed by the National Electrical Manufacturers Association (NEMA) in conjunction with the American College of Radiology (ACR). DICOM images can contain patient information, as well as information about the image itself such as the modality (e.g., CT, MRI, ultrasound, etc.), image dimensions, the date the image was taken, and much more.

   In addition to the imaging data, DICOM files often contain a large amount of metadata. This metadata includes details about the patient (like name, age, and sex), details about the equipment used to capture the image, and specifics about how the image was captured (like the angle, duration of exposure, etc.).

A key feature of the DICOM format is that it allows for patient information to be linked with the image data. This linking allows for better organization and retrieval of the images in a healthcare setting. This information can be extremely beneficial in a clinical workflow, where multiple medical professionals may need to access the image and understand the context in which it was taken.

One important thing to note is that while the DICOM standard provides a common format for medical images, the images themselves can vary widely. Differences in the equipment used, the settings selected by the technician, and even the specific patient can all result in differences in the images. As such, working with DICOM images can sometimes be a complex task.

a. Since the images are stored as DICOM files, they need to be converted into a more usable format (e.g., arrays of pixel intensities). Also, since the pixel intensities might vary across different images, we need to normalize these values.
b. It was observed that the if all the images are loaded in memory, the kernel usually crashes! Be it on PC, Kaggle or Google Colab. Hence it was decided to explore suitable methods which will enable training of model on all images without exhausting resources. The efforts are elaborated in Attempt 1 & Attemp 2 below. Attempt1 tries to train the model on subset of images in DICOM format. In Attempt 2 DICOM images are converted to PNG format are placed suitably to work with flow from directory function. Conclusion of the same shall be noted in observation below.
c. In either case to streamline and explore the solutions, metadata dataframe has been created which will collect data from class info, train labels csv files as well as data from the DICOM images. The metadata shall be saved in csv format. The dataframe shall have 1 entry for each unique image. It will hold available informaiton for both train and test images, list of bounding boxes, labels as per Target (2) and Class info (3). Further each image shall be assigned a set name, either training, validation or test set. To enable easy slicing of the dataframe.
d. Visualization: We'll display a few images from each class ('Lung Opacity', 'No Lung Opacity / Not Normal', 'Normal') to get an understanding of what each class looks like.

Code snippets- Data pre-processing

(A) - Function to join all strings in the list to text

```
# Define helper functions usefule in creation of metadata dataframe

# Function to join all strings in the list [use for converting list in a column to text]
def join_strings(class_info):
    if class_info is not None:
        return ' '.join(class_info)
    return None

# Function to map 'class_info_text' to a label
def map_to_label(class_info_text):
    if class_info_text == 'Normal':
        return 0
    elif class_info_text == 'No Lung Opacity / Not Normal':
        return 1
    elif class_info_text is None:
        return None
    else:
        return 2
```

(B) Creation of Metadata

The objective is to have all data to be available in one dataframe. Logic maintained as to have unique records for each train and test images. In case image has multiple bounding boxes then the list of all the boxes will be in dataframe. This metadata will also hold test images. This metadata also hold additional information from dicom images shown below as header information. Out of total available information patient age, sex and other explorable information will be utilized for model building.

```
# let's see the header of typical dicom image
train_images[0]
```

```
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: 1.2.276.0.7230010.3.1.4.8323329.285
30.1517874485.775526
(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: 1.2.276.0.7230010.3.1.4.8323329.285
30.1517874485.775526
(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-63a80b6bdd
d6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bdd
d6'
(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: 1.2.276.0.7230010.3.1.2.8323329.285
30.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.285
30.1517874485.775524
(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, 0.14300000000
000002]
(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
```

Before creating the metadata, the images are converted to png format. Processing with huge number of dicom images resulting in crash of notebook or kernel crash.

```
# function for Coverting images to png format and save them to folders suitable
for flow_from _directory funciton
def convert_save(filename, ds, n_classes, label, dataset, img_dims=(image_size,
image_size), color='L'):
    #ds = dcmread(os.path.join(source_folder, filename + '.dcm'))  # Assuming
filenames are without .dcm extension

    # Convert to grayscale, resize, and normalize pixel values
    img = Image.fromarray(ds.pixel_array).convert(color)

    # Resize the image
    img = img.resize(img_dims)

    # Standardize the image (for both grayscale and RGB)
    img = (img - np.min(img)) / 255.  #(np.max(img) - np.min(img))

    # Decide where to put th conveted image based on label and which set it
belongs to...
    # This way we can access the images using flow from directory function
    if n_classes == 2:
        if label is not None:  # 1.0 corresponds to 'Opacity'
            class_dir = os.path.join(train_png_images_path if dataset ==
'train' else val_png_images_path, lbl2[label])
        else:
            class_dir = os.path.join(test_png_images_path)
    else:
        if label is not None:  # 1.0 corresponds to 'Opacity'
            class_dir = os.path.join(train_png_images_path if dataset ==
'train' else val_png_images_path, lbl3[label])
        else:
            class_dir = os.path.join(test_png_images_path)

    # Create class directories if they don't exist
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)

    # Save as PNG
    Image.fromarray((img * 255).astype(np.uint8)).save(os.path.join(class_dir,
filename + '.png'))
    # Saving with original filename

    return class_dir
```

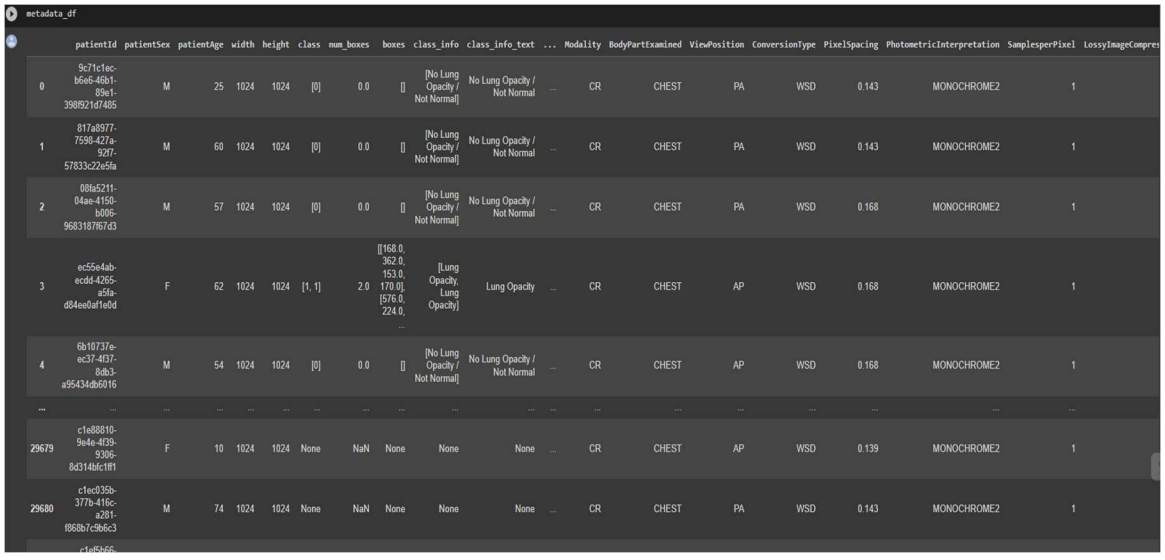
Below code snippet will create metadata as we are looking for

```
val_ratio=0.2

# Initialize ViewPosition column if it doesn't exist
if 'ViewPosition' not in train_labels.columns:
    train_labels['ViewPosition'] = np.nan

# Prepare the metadata dictionary
metadata = {'patientId': [], 'patientSex': [], 'patientAge': [], 'width': [],
'height': [],
            'class': [], 'num_boxes': [], 'boxes': [], 'class_info': [],
'class_info_text': [],
            'label2': [], 'label3': [], 'set_name': [], 'folder': [],
'Modality': [],
```

```
            'BodyPartExamined': [], 'ViewPosition': [], 'ConversionType': [],
'PixelSpacing': [],
            'PhotometricInterpretation': [], 'SamplesperPixel': [],
'LossyImageCompression': [],
            'LossyImageCompressionMethod': [], 'PixelRepresentation': [] }

# Assuming train_labels is your DataFrame
unique_train_labels = train_labels.drop_duplicates(subset='patientId',
keep='first')

# Now, 'X' will be unique 'patientId's
train_filenames = unique_train_labels['patientId']

# And 'y' will be corresponding 'Target'
labels = unique_train_labels['Target']

test_filenames = test_labels['patientId']

# Split filenames into training and validation sets
train_files, val_files, train_lbls, val_lbls = train_test_split(
        train_filenames, labels, stratify=labels, test_size=val_ratio,
random_state = 42)

for set_name, image_list in [('train', train_files), ('val', val_files),
('test', test_filenames)]:
    for filename in tqdm(image_list, desc=set_name):
        img_path = train_images_path if set_name != 'test' else
test_images_path
        image = dcmread(os.path.join(img_path, filename + '.dcm'))

        # Extract width, height and number of color channels (=1 as they are
all grayscale)
        height, width = image.Rows, image.Columns

        # Append the computed metadata to the lists in the dictionary
        metadata['patientId'].append(image.PatientID)
        metadata['patientSex'].append(image.PatientSex)
        metadata['patientAge'].append(image.PatientAge)
        metadata['width'].append(height)
        metadata['height'].append(width)

        metadata['Modality'].append(image.Modality)

metadata['LossyImageCompression'].append(pd.to_numeric(image.LossyImageCompress
ion))

metadata['LossyImageCompressionMethod'].append(image.LossyImageCompressionMetho
d)
        metadata['BodyPartExamined'].append(image.BodyPartExamined)
        metadata['ViewPosition'].append(image.ViewPosition)
        metadata['ConversionType'].append(image.ConversionType)
        metadata['SamplesperPixel'].append(image.SamplesPerPixel)

metadata['PhotometricInterpretation'].append(image.PhotometricInterpretation)

metadata['PixelSpacing'].append(str.format("{:4.3f}",image.PixelSpacing[0]))
        metadata['PixelRepresentation'].append(image.PixelRepresentation)

        # Assign the class label and the number of bounding boxes
        if set_name != 'test':
            class_label = train_class_dict[image.PatientID]
            class_inf = train_class_info_dict[image.PatientID]
            class_inf_text = join_strings(list(set(list(class_inf))))
```

```
            num_boxes = len(train_anno_dict.get(image.PatientID, []))
            boxes = train_anno_dict.get(image.PatientID, [])
            label2 = class_label[0]
            label3 = map_to_label(class_inf_text)

            # Update 'ViewPosition' in train_labels only when processing
training images
            train_labels.loc[train_labels['patientId']== image.PatientID,
'ViewPosition'] = image.ViewPosition
        else:
            # For test images, we don't have the class label and the bounding
boxes
            class_label = None
            class_inf = None
            class_inf_text = None
            num_boxes = None
            label2 = None
            label3 = None
            boxes = None

        metadata['class'].append(class_label)
        metadata['num_boxes'].append(num_boxes)
        metadata['boxes'].append(boxes)
        metadata['class_info'].append(class_inf)
        metadata['class_info_text'].append(class_inf_text)
        metadata['label2'].append(label2)
        metadata['label3'].append(label3)
        metadata['set_name'].append(set_name)
        if n_classes == 2:
            metadata['folder'].append(convert_save(filename, image, n_classes,
label2, set_name,
                                                   img_dims=(image_size,
image_size), color=set_color))
        else:
            metadata['folder'].append(convert_save(filename, image, n_classes,
label3, set_name,
                                                   img_dims=(image_size,
image_size), color=set_color))

# Convert the dictionary to a pandas DataFrame
metadata_df = pd.DataFrame(metadata)
```

Below screenshot image is as output of newly created metadata for our reference



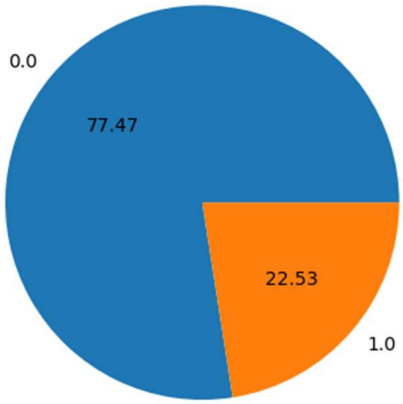
metadata_df

	patientId	patientSex	patientAge	width	height	class	num_boxes	boxes	class_info	class_info_text	...	Modality	BodyPartExamined	ViewPosition	ConversionType	PixelSpacing	PhotometricInterpretation	SamplesperPixel	LossyImageCompres
0	9c71c1ec-b6e6-46b1-89e1-398f921d7485	M	25	1024	1024	[0]	0.0	[]	[No Lung Opacity / Not Normal]	No Lung Opacity / Not Normal	...	CR	CHEST	PA	WSD	0.143	MONOCHROME2	1	
1	817a8977-7598-427a-92f7-57833c22e5fa	M	60	1024	1024	[0]	0.0	[]	[No Lung Opacity / Not Normal]	No Lung Opacity / Not Normal	...	CR	CHEST	PA	WSD	0.143	MONOCHROME2	1	
2	08fa5211-04ae-4150-b006-968318f67d3	M	57	1024	1024	[0]	0.0	[]	[No Lung Opacity / Not Normal]	No Lung Opacity / Not Normal	...	CR	CHEST	PA	WSD	0.168	MONOCHROME2	1	
3	ec55e4ab-ecdd-4265-a5fa-d84ee0af1e0d	F	62	1024	1024	[1, 1]	2.0	[[168.0, 362.0, 153.0, 170.0], [576.0, 224.0, ...	[Lung Opacity, Lung Opacity]	Lung Opacity	...	CR	CHEST	AP	WSD	0.168	MONOCHROME2	1	
4	6b10737e-ec37-4f37-8db3-a95434db6016	M	54	1024	1024	[0]	0.0	[]	[No Lung Opacity / Not Normal]	No Lung Opacity / Not Normal	...	CR	CHEST	AP	WSD	0.168	MONOCHROME2	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29679	c1e88810-9e4e-4f39-9306-8d314bfc1ff1	F	10	1024	1024	None	NaN	None	None	None	...	CR	CHEST	AP	WSD	0.139	MONOCHROME2	1	
29680	c1ec035b-377b-416c-a281-f868b7c9b6c3	M	74	1024	1024	None	NaN	None	None	None	...	CR	CHEST	PA	WSD	0.143	MONOCHROME2	1	
	c1ef5b66-																		

6. Exploratory Data Analysi (EDA)

In statistics, exploratory data analysis (EDA) is an approach of analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling and thereby contrasts traditional hypothesis testing. Exploratory data analysis has been to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments.

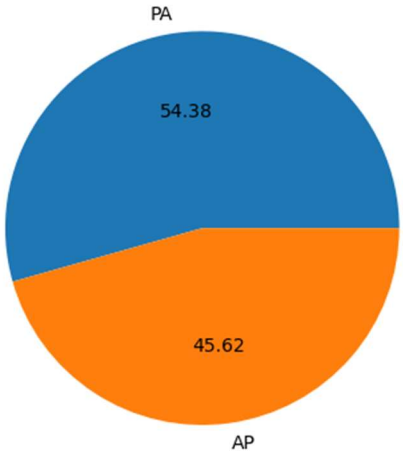
(a) Unique target classes in set and their distribution. – 23% images are with lung opacity and rest 77% are normal x-rays.



(b) Visualizing positions of x-ray images -

```
metadata_train['ViewPosition'].value_counts()
```

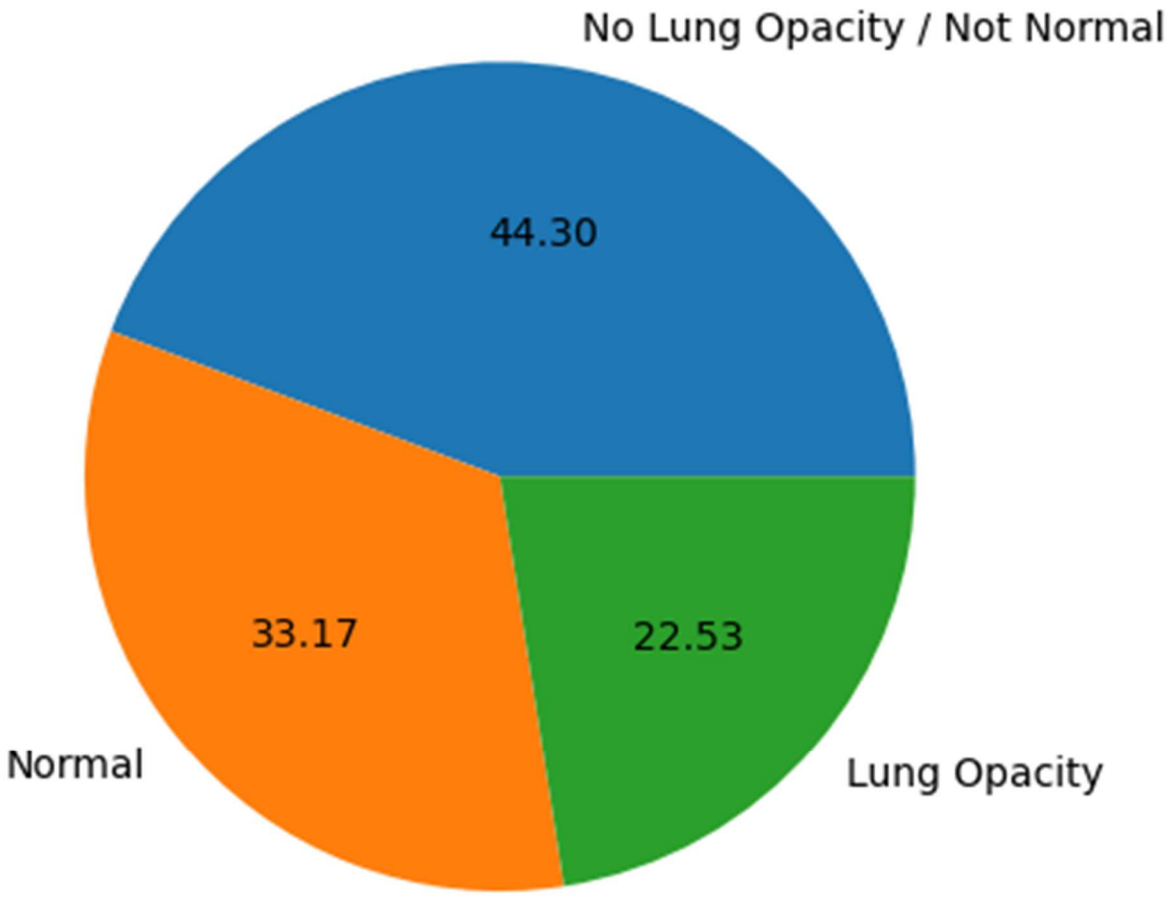
```
PA    14511
AP    12173
Name: ViewPosition, dtype: int64
```



(c) Visualizing class target distribution

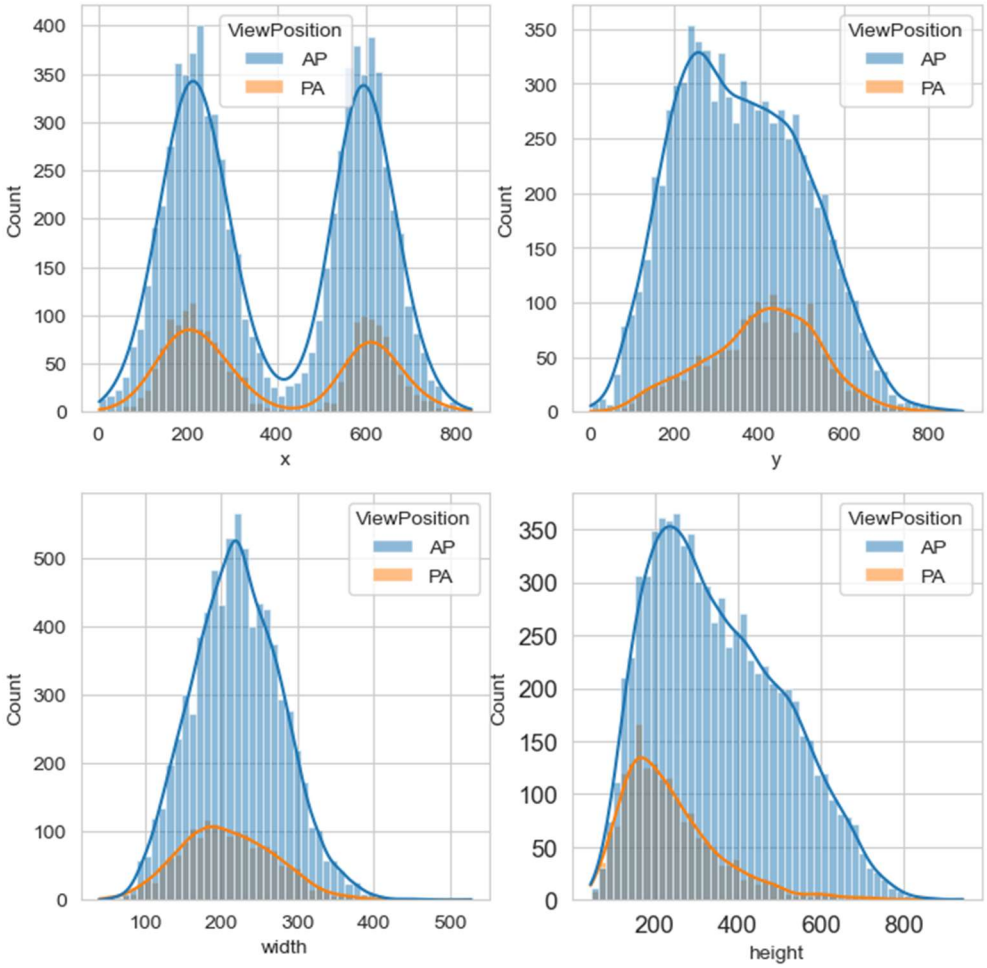
```
metadata_train['class_info_text'].value_counts()
```

```
No Lung Opacity / Not Normal    11821
Normal                           8851
Lung Opacity                     6012
Name: class_info_text, dtype: int64
```



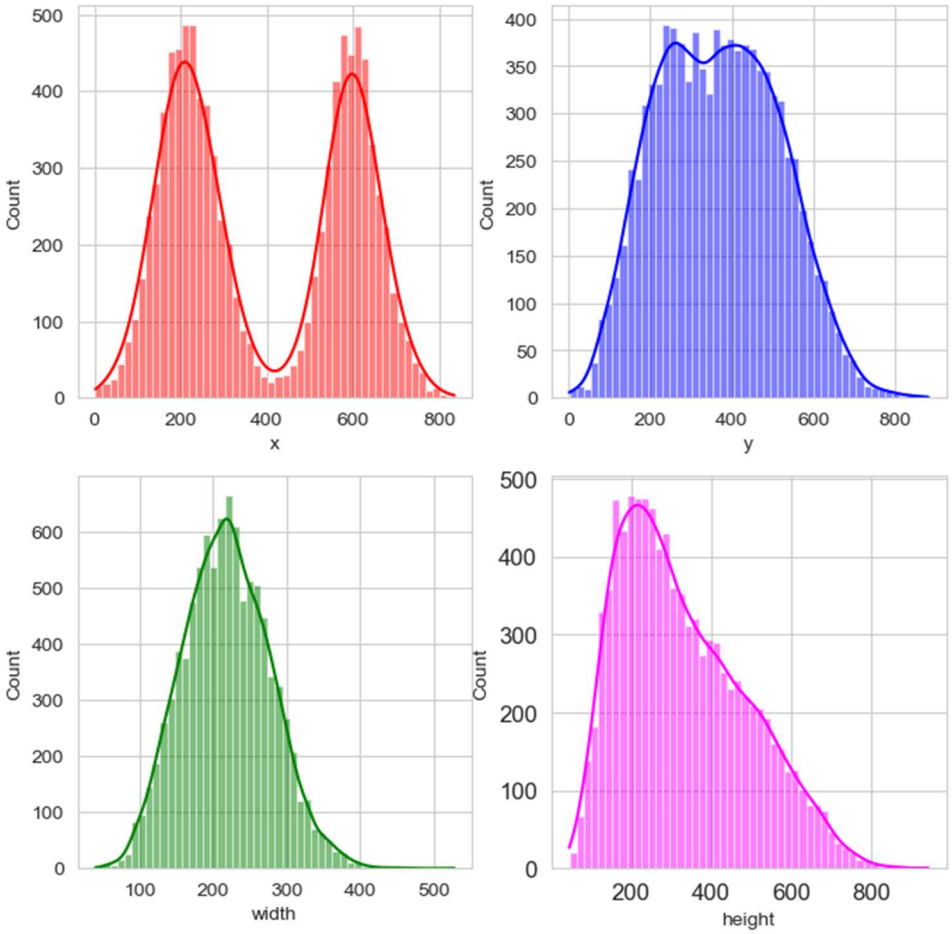
(d) Detection of Lung Opacity Window

```
import matplotlib.pyplot as plt
import seaborn as sns

# Select only data points with Target==1
target1 = train_labels[train_labels['Target'] == 1]

# Set style for seaborn plots
sns.set_style('whitegrid')

# Create subplots
fig, ax = plt.subplots(2, 2, figsize=(8, 8))

# Create a histogram for each variable of interest
sns.histplot(data=target1, x='x', kde=True, bins=50, hue="ViewPosition", ax=ax[0, 0])
sns.histplot(data=target1, x='y', kde=True, bins=50, hue="ViewPosition", ax=ax[0, 1])
sns.histplot(data=target1, x='width', kde=True, bins=50, hue="ViewPosition", ax=ax[1, 0]
sns.histplot(data=target1, x='height', kde=True, bins=50, hue="ViewPosition", ax=ax[1, 1

# Modify tick parameters
plt.tick_params(axis='both', which='major', labelsize=12)

# Display the plot
plt.show()
```



```
import matplotlib.pyplot as plt
import seaborn as sns

# Select only data points with Target==1
target1 = train_labels[train_labels['Target'] == 1]

# Set style for seaborn plots
#sns.set_style('whitegrid')

# Create subplots
fig, ax = plt.subplots(2, 2, figsize=(8, 8))

# Create a histogram for each variable of interest
sns.histplot(data=target1, x='x', kde=True, bins=50, color="red", ax=ax[0, 0])
sns.histplot(data=target1, x='y', kde=True, bins=50, color="blue", ax=ax[0, 1])
sns.histplot(data=target1, x='width', kde=True, bins=50, color="green", ax=ax[1, 0])
sns.histplot(data=target1, x='height', kde=True, bins=50, color="magenta", ax=ax[1, 1])

# Modify tick parameters
plt.tick_params(axis='both', which='major', labelsize=12)
```

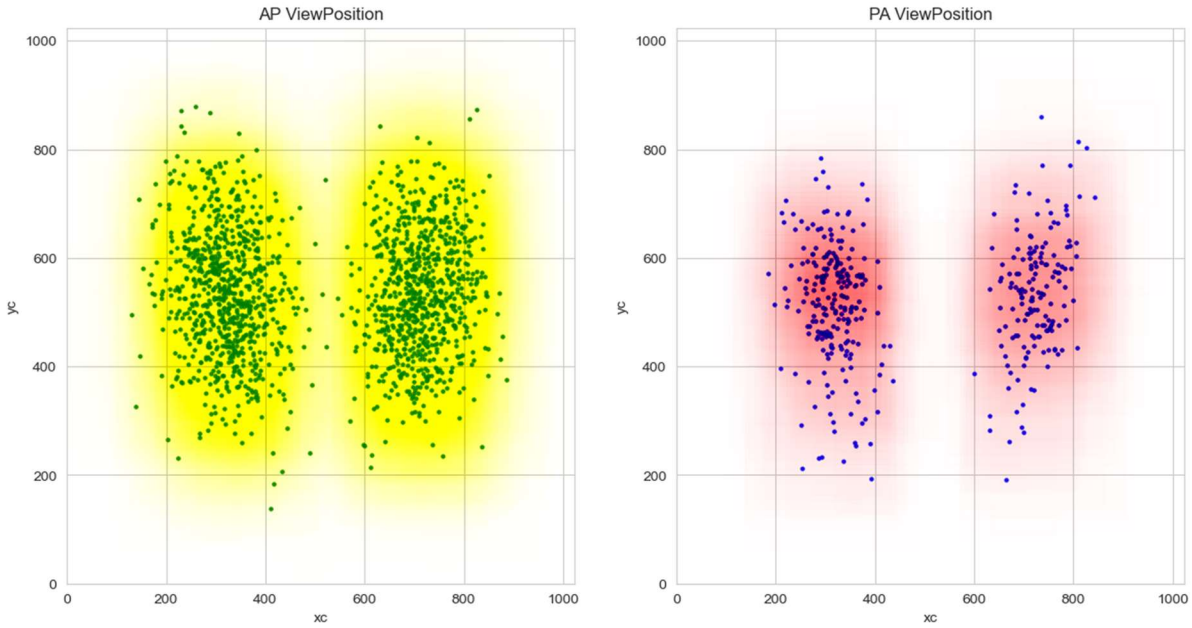
```
# Display the plot
plt.show()
```



With the help of Kevin's Kernel, the rectangles are created. A sample of center points superposed with the corresponding rectangles. Below code snippet and charts will provide additional information as

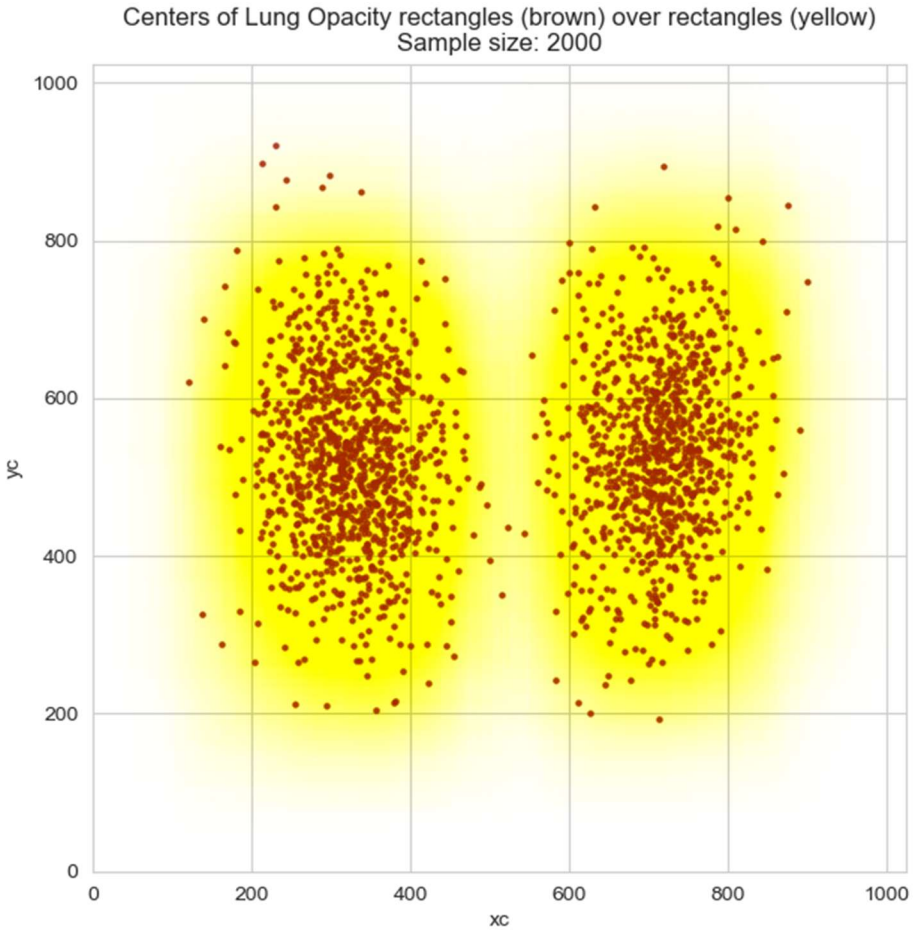
```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

# Sample 5000 points from the target
target_sample = target1.sample(2000)

# Calculate the center of the rectangles
target_sample['xc'] = target_sample['x'] + target_sample['width'] / 2
target_sample['yc'] = target_sample['y'] + target_sample['height'] / 2
```

```
# Split the samples by ViewPosition
target_sample_AP = target_sample[target_sample['ViewPosition'] == 'AP']
target_sample_PA = target_sample[target_sample['ViewPosition'] == 'PA']

# Create a new figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(14, 7))

# Plot the samples with ViewPosition == 'AP'
axs[0].set_title("AP ViewPosition")
target_sample_AP.plot.scatter(x='xc', y='yc', xlim=(0, 1024), ylim=(0, 1024), ax=axs[0],
for i, crt_sample in target_sample_AP.iterrows():
    axs[0].add_patch(Rectangle(xy=(crt_sample['x'], crt_sample['y']),
                               width=crt_sample['width'],
                               height=crt_sample['height'],
                               alpha=3.5e-3, color="yellow"))

# Plot the samples with ViewPosition == 'PA'
axs[1].set_title("PA ViewPosition")
target_sample_PA.plot.scatter(x='xc', y='yc', xlim=(0, 1024), ylim=(0, 1024), ax=axs[1],
for i, crt_sample in target_sample_PA.iterrows():
    axs[1].add_patch(Rectangle(xy=(crt_sample['x'], crt_sample['y']),
                               width=crt_sample['width'],
                               height=crt_sample['height'],
                               alpha=3.5e-3, color="red"))

# Display the plot
plt.show()
```



```python
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

# Create a new figure and a new subplot
fig, ax = plt.subplots(1, 1, figsize=(7, 7))

# Sample 2000 points from the target
target_sample = target1.sample(2000)
```

```python
# Calculate the center of the rectangles
target_sample['xc'] = target_sample['x'] + target_sample['width'] / 2
target_sample['yc'] = target_sample['y'] + target_sample['height'] / 2

# Set the title of the plot
plt.title("Centers of Lung Opacity rectangles (brown) over rectangles (yellow)\nSample s

# Plot the centers of the rectangles
target_sample.plot.scatter(x='xc', y='yc', xlim=(0, 1024), ylim=(0, 1024), ax=ax, alpha=

# Draw the rectangles
for i, crt_sample in target_sample.iterrows():
    ax.add_patch(Rectangle(xy=(crt_sample['x'], crt_sample['y']),
                           width=crt_sample['width'],
                           height=crt_sample['height'],
                           alpha=3.5e-3, color="yellow"))

# Display the plot
plt.show()
```



Observations from the above charts as

1. There is significant imbalance in consideration of binary classification
2. However, for 3 classification, the imbalance is tolerable
3. Detected lung opacity window has different distribution based on view position.

7. Data Analysis

Isolating columns from metadata dataframe which are with similar meaning. Then checking how the data looks like as

```
# Isolate columns where ever entry is a list for studying the columns of Metadata dataframe
columns = list(metadata_df.columns)

remove_columns = ['class', 'boxes', 'class_info']

columns = [col for col in columns if col not in remove_columns]
```

```
[ ] nx = studydf(metadata_df[columns])
```

```
Data set shape: Rows = 29684,   Columns = 21

Top & Bottom 5 Rows of the Dataset
```

	patientId	patientSex	patientAge	width	height	num_boxes	class_info_text	label2	label3	set_name	...	Modality	BodyPartExamined	ViewPosition	ConversionType	PixelSpacing	PhotometricInterpretation
0	9c71c1ec-b6e6-46b1-89e1-398f921d7485	M	25	1024	1024	0.0	No Lung Opacity / Not Normal	0.0	1.0	train	...	CR	CHEST	PA	WSD	0.143	MONOCHROME2
1	817a8977-7598-427a-92f7-57833c22e5fa	M	60	1024	1024	0.0	No Lung Opacity / Not Normal	0.0	1.0	train	...	CR	CHEST	PA	WSD	0.143	MONOCHROME2
2	08fa5211-04ae-4150-b006-9683187f67d3	M	57	1024	1024	0.0	No Lung Opacity / Not Normal	0.0	1.0	train	...	CR	CHEST	PA	WSD	0.168	MONOCHROME2
3	ec55e4ab-ecdd-4265-a5fa-d84ee0af1e0d	F	62	1024	1024	2.0	Lung Opacity	1.0	2.0	train	...	CR	CHEST	AP	WSD	0.168	MONOCHROME2

There are 3000 records which has missing values under num_boxes, class_info_text, label2 and label3.

```
Meta data of the dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29684 entries, 0 to 29683
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   patientId                   29684 non-null  object
 1   patientSex                  29684 non-null  object
 2   patientAge                  29684 non-null  object
 3   width                       29684 non-null  int64
 4   height                      29684 non-null  int64
 5   num_boxes                   26684 non-null  float64
 6   class_info_text             26684 non-null  object
 7   label2                      26684 non-null  float64
 8   label3                      26684 non-null  float64
 9   set_name                    29684 non-null  object
 10  folder                      29684 non-null  object
 11  Modality                    29684 non-null  object
 12  BodyPartExamined            29684 non-null  object
 13  ViewPosition                29684 non-null  object
 14  ConversionType              29684 non-null  object
 15  PixelSpacing                29684 non-null  object
 16  PhotometricInterpretation   29684 non-null  object
 17  SamplesperPixel             29684 non-null  int64
 18  LossyImageCompression       29684 non-null  int64
 19  LossyImageCompressionMethod 29684 non-null  object
 20  PixelRepresentation         29684 non-null  int64
dtypes: float64(3), int64(5), object(13)
memory usage: 4.8+ MB

3000 missing values detected
```

Studying of required columns within metadata helps us to have broader idea as columns are shaped.

```
studyAllCols(metadata_df[columns])
```

	CatFeature	ClassCount	DType	ClassUniqueValues	ClassUniqueValueCount	Li
0	patientId	29684	object	[9c71c1ec-b6e6-46b1-89e1-398f921d7485, 817a897...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	
1	patientSex	2	object	[M, F]	[16880, 12804]	
2	patientAge	98	int64	[25, 60, 57, 62, 54, 72, 33, 13, 51, 22, 49, 5...	[346, 642, 695, 541, 724, 274, 427, 115, 643, ...	
3	width	1	int64	[1024]	[29684]	
4	height	1	int64	[1024]	[29684]	
5	num_boxes	5	float64	[0.0, 2.0, 1.0, 3.0, 4.0]	[20672, 3266, 2614, 119, 13]	
6	class_info_text	3	object	[No Lung Opacity / Not Normal, Lung Opacity, N...	[11821, 6012, 8851]	
7	label2	2	float64	[0.0, 1.0]	[20672, 6012]	
8	label3	3	float64	[1.0, 2.0, 0.0]	[11821, 6012, 8851]	
9	set_name	3	object	[train, val, test]	[21347, 5337, 3000]	
10	folder	5	object	[train_images224RGB2\No_Opacity, train_images2...	[16537, 4810, 4135, 1202, 3000]	
11	Modality	1	object	[CR]	[29684]	
12	BodyPartExamined	1	object	[CHEST]	[29684]	
13	ViewPosition	2	object	[PA, AP]	[16129, 13555]	
14	ConversionType	1	object	[WSD]	[29684]	
15	PixelSpacing	7	object	[0.143, 0.168, 0.139, 0.171, 0.194, 0.115, 0.199]	[9979, 9829, 6118, 2221, 1524, 10, 3]	
16	PhotometricInterpretation	1	object	[MONOCHROME2]	[29684]	
17	SamplesperPixel	1	int64	[1]	[29684]	
18	LossyImageCompression	1	int64	[1]	[29684]	
19	LossyImageCompressionMethod	1	object	[ISO_10918_1]	[29684]	
20	PixelRepresentation	1	int64	[0]	[29684]	

Checking of statistical aspects of data as

```
desc(metadata_df).T
```

	count	mean	std	min	25%	50%	75%	max	cv	Ske
patientAge	29684.0	47.032206	16.951208	1.0	34.0	49.0	59.0	412.0	0.360417	-0.3482
width	29684.0	1024.000000	0.000000	1024.0	1024.0	1024.0	1024.0	1024.0	0.000000	Na
height	29684.0	1024.000000	0.000000	1024.0	1024.0	1024.0	1024.0	1024.0	0.000000	Na
num_boxes	26684.0	0.358080	0.712231	0.0	0.0	0.0	0.0	4.0	1.989028	1.5082
label2	26684.0	0.225304	0.417790	0.0	0.0	0.0	0.0	1.0	1.854342	1.6178
label3	26684.0	0.893607	0.738716	0.0	0.0	1.0	1.0	2.0	0.826668	-0.4320
SamplesperPixel	29684.0	1.000000	0.000000	1.0	1.0	1.0	1.0	1.0	0.000000	Na
LossyImageCompression	29684.0	1.000000	0.000000	1.0	1.0	1.0	1.0	1.0	0.000000	Na
PixelRepresentation	29684.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	NaN	Na

Observations

- There are 29684 records / images and all are with same size.
- There are no duplicate records.
- There are none to four bounding boxes are present in x-rays.
- Max patientAge of 412 is wrongly maintained.
- From all these observations, it is evident that metadata is created properly for next steps towards model building.
- All images have single Modality, Body Part Examined, ConversionType, SamplesperPixel, LossyImageCompression, PixelRepresentation, LossyImageCompressionMethod and PhotometricInterpretation as indicated above
- ViewPosition have two value AP (Anterior/Posterior) and PA (Posterior/Anterior), as some 13555 chest xrays are taken from front AP and some 16129 from back PA
- PixelSpacing has 7 unique values, which will need to be accounted for while conversion and resizing

Discussion of DICOM images metadata for the images in our Datasets

For detailed analysis required for model building, the other parameters like below can be utilized as,

Photometric Interpretation - Monochrome2 pixel data refers to a specific data format used in medical imaging. In medical imaging, pixels are the smallest units of information that make up an image. Monochrome2 pixel data specifically refers to grayscale images, where each pixel value represents a different shade of gray.

The "2" in Monochrome2 represents the ordering of pixel values, with the darkest pixels having the lowest values and the lightest pixels having the highest values. This ordering convention is used to simplify image processing algorithms and enhance visualization.

Monochrome2 pixel data is commonly used in medical imaging to represent X-rays, CT scans, and other types of grayscale medical images. It is a widely accepted format within the medical imaging community.

The minimum sample value is intended to be displayed as black after any VOI gray scale transformations have been performed. See PS3.4. This value may be used only when Samples per Pixel (0028,0002) has a value of 1. May be used for pixel data in a Native (uncompressed) or Encapsulated (compressed) format.

Lossy Image Compression - Lossy compression is a technique used in image processing to reduce the file size of an image by discarding some of the image data. The main aim of lossy compression is to achieve a smaller file size while tolerating an acceptable level of quality loss.

In lossy compression, the image is divided into small blocks of pixels, and then various techniques are applied to eliminate redundant or irrelevant information in those blocks. This removal of data results in a reduction in file size. However, since the compression involves discarding information, there is a loss in image quality compared to the original uncompressed image.

Lossy compression is particularly useful when the storage or bandwidth is limited because it enables the transmission or storage.

Lossy Image Compression Method - Lossy image compression is a method used to reduce the size of image files by selectively discarding certain image information that is perceived as less important or less noticeable to the human eye. In the context of X-ray images in ISO_10918_1, lossy image compression aims to reduce the file size of these images while maintaining an acceptable level of diagnostic quality.

Pixel Spacing - Pixel spacing refers to the physical distance between individual pixels on a display or an imaging device. It is typically measured in millimeters (mm) and determines the level of detail and clarity that can be achieved in an image.

In a display, such as a computer monitor or a smartphone screen, pixel spacing refers to the distance between adjacent pixels. A smaller pixel spacing indicates a higher pixel density and results in a crisper and more detailed image.

In medical imaging, pixel spacing refers to the physical distance between pixels in a scanned image or a digital x-ray. This measurement is essential for accurately representing the size of objects or anatomical structures being imaged.

Larger pixel spacing can indeed result in a less sharp image quality. When pixels are spaced further apart, there is a decrease in pixel density, which means that each pixel is responsible for capturing a larger amount of information. Consequently, this can lead to a loss of detail and clarity in the final image.

Pixel Representation - It refers to the manner in which digital images are encoded and stored in computer systems. A pixel, short for picture element, is the smallest unit of a digital image. It represents a single point in an image and contains information about its color and brightness.

In pixel representation, each pixel is assigned a numerical value that represents its color. This value is typically a combination of red, green, and blue (RGB) values, which determine the intensity of these three primary colors for that particular pixel. The RGB values can range from 0 to 255, allowing for millions of different color combinations.

Pixels are arranged in a grid-like pattern to form a complete image.

Sample per Pixel – Sample per Pixel refers to the number of samples taken for each pixel in an image. Each pixel in a digital image is made up of a combination of different color channels, such as red, green, and blue. The sample per pixel value determines the level of detail and color accuracy in the image.

In images with a lower sample per pixel value, there are fewer samples taken for each pixel, resulting in lower detail and potentially less accurate color representation. On the other hand, images with a higher sample per pixel value have more samples taken for each pixel, resulting in higher detail and more accurate color representation.

The sample per pixel value is typically expressed as a power of 2, such as 1. One sample per pixel means that each composite pixel value is identical to the single Pixel Data value

Modality - Modality in medical imaging refers to the different technologies and techniques used to produce images of the human body for diagnostic purposes. It includes various types of imaging equipment, such as X-rays, ultrasounds, computed tomography (CT), magnetic resonance imaging (MRI), and nuclear medicine scans. Each modality has its own unique way of capturing images and provides different types of information about the body.

CR Computed Radiography Type of device, process or method that originally acquired the data used to create the Instances in this series.

Body Part Examined - Chest Text description of the part of the body examined.

View Position - The view position in an X-ray refers to the specific position of the patient's body or the part being imaged in relation to the X-ray machine and the film or digital imaging receptor. Different view positions are used to obtain the best possible images of different body parts. It has typically two values as AP (Anterior / Posterior) and PA (Posterior / Anterior).

Conversion Type - Conversion type in image processing refers to the process of converting an image from one format to another. Image formats such as JPEG, PNG, and GIF have different properties, such as file size, color depth, and compression ratio. In some cases, it may be necessary to convert an image to a different format in order to optimize it for a specific application or to meet certain requirements.

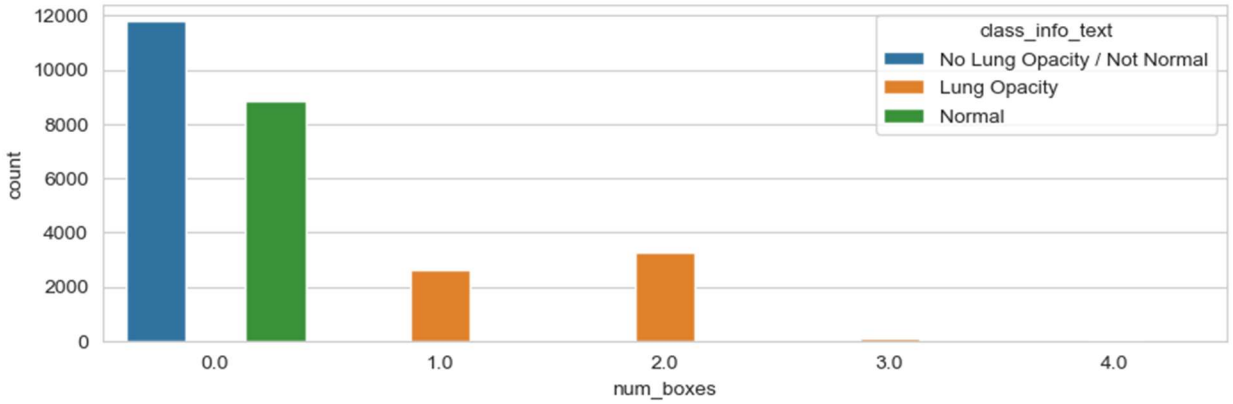
In image processing, a conversion type refers to the specific format or type of file being used for image data. This type of file determines how the image data is stored and interpreted by the computer.

One common conversion type in image processing is the WordStar Document (WSD) file format. This format was originally developed for word processing, but it can also be used to store image data in a specific way.

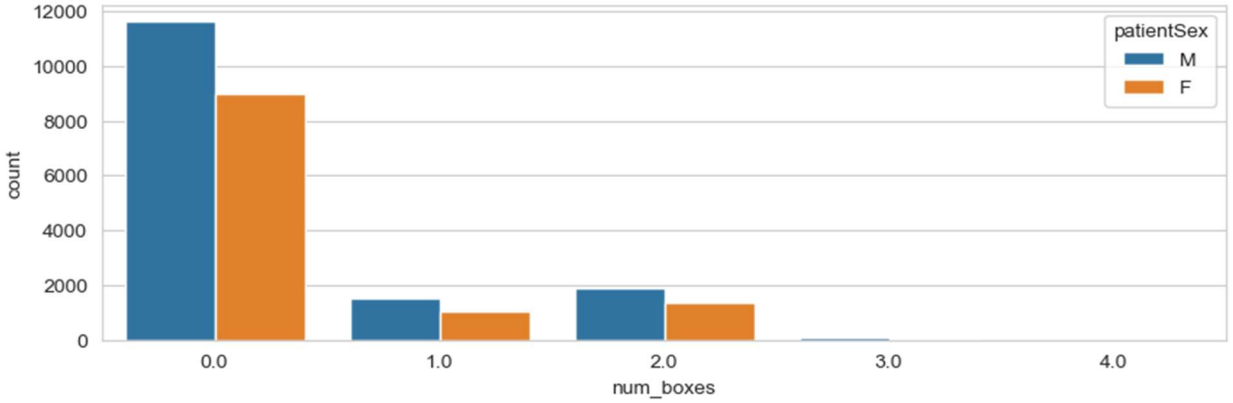
Exploring more on EDA -

Analyzing EDA with respect to visualization is always appealing and visual trends are easy and fast to understand. We can make the same inferences by looking generated graphs /charts.

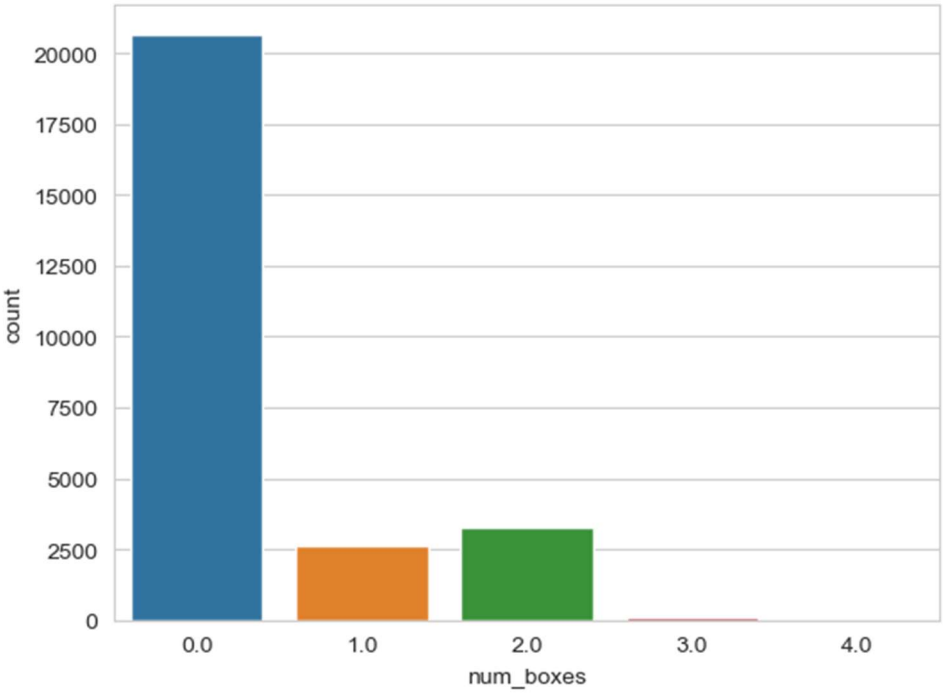
A. - Distribution of bounding boxes when x-rays are classied with class_info_text



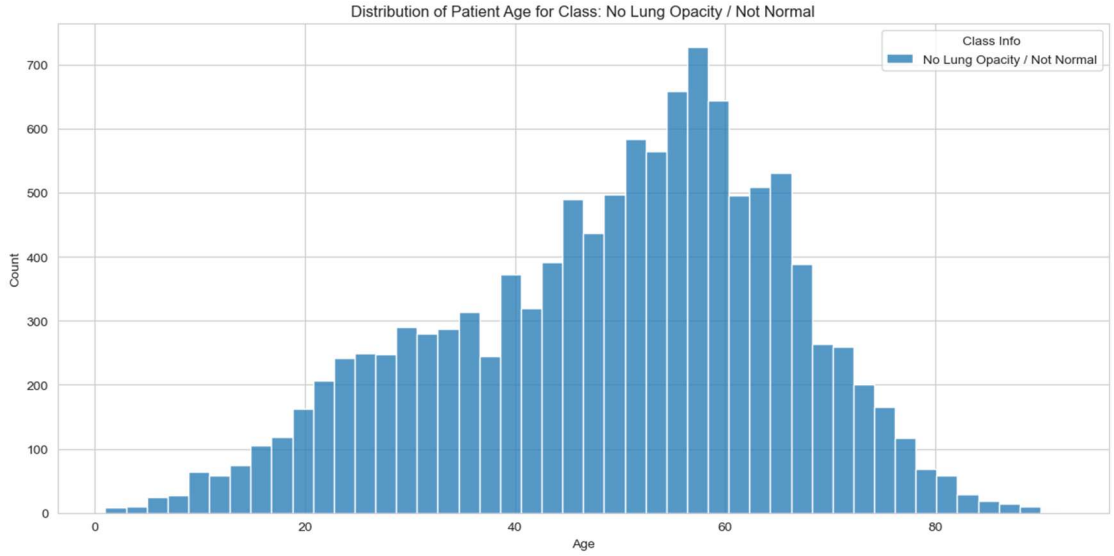
B. How Male and Female (as patientSex) is distributed across number of bounding boxes

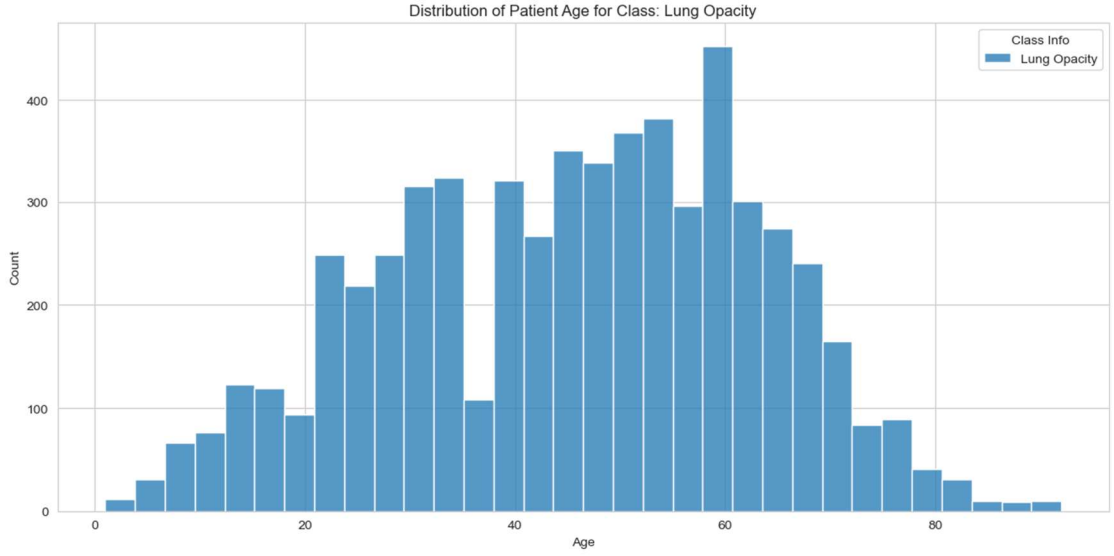


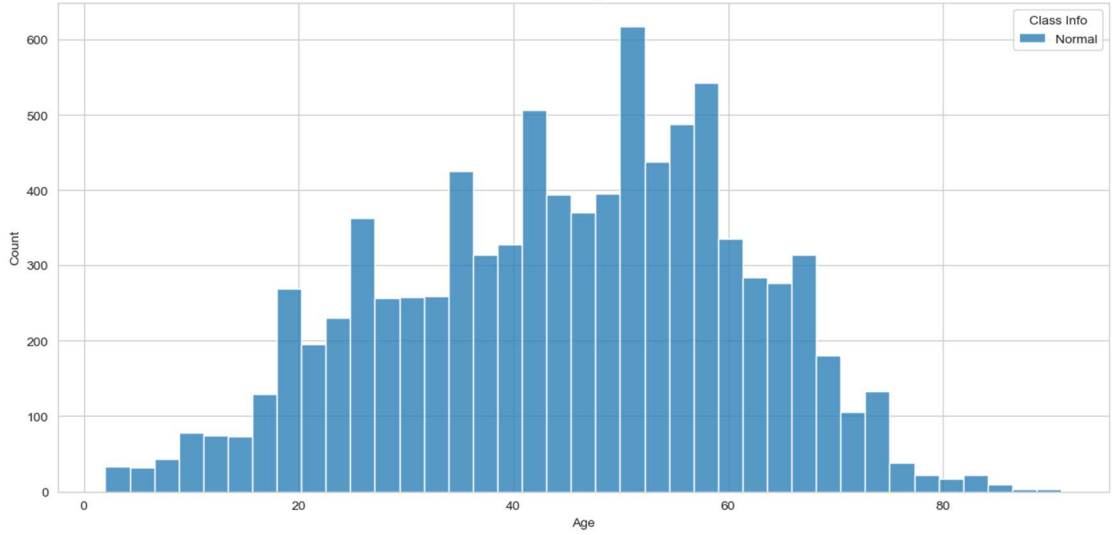
C. See how bounding boxes are against each x-ray



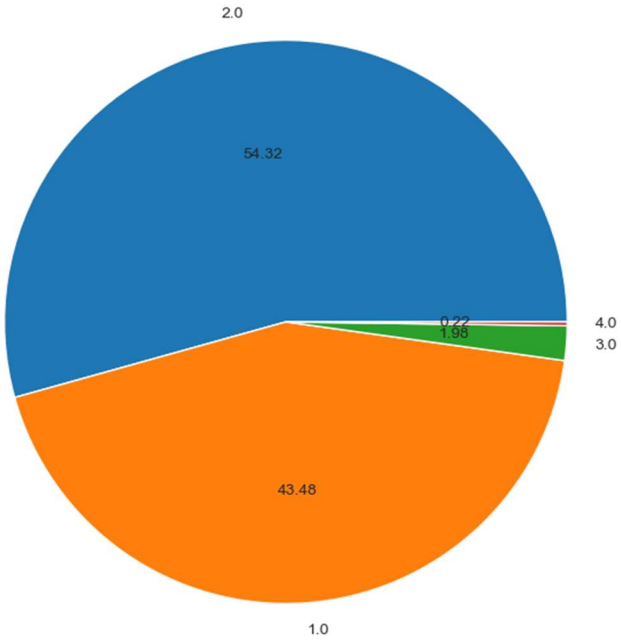
D. Distribution of Patient Age for Class : No Lung Opacity / Not Normal







E. Distribution of number of bounding boxes in case of Lung opacity images



Observations from above Charts

1. Bounding boxes are specified only in the case of Lung opacity
2. Males are more likely to have their chest x-ray taken than females
3. Upto 4 bounding boxes can be there if the diagnosis is 'Lung Opacity'
4. No bounding boxes are specified for other cases
5. Roughly (23%) Images are with 'Lung Opacity'
6. Hence class imbalance is tolerable with respect to othe classes.
7. All images are 1024x1024 grey scale images
8. Male cases slightly dominate the female ones
9. Distribution of ages for show negative skew to the left. every class Peaks at the age of 55-60
10. Some Age values are wrong. But since we are not using this info for training, we can ignore the same.

Display of DICOM images with bounding boxes

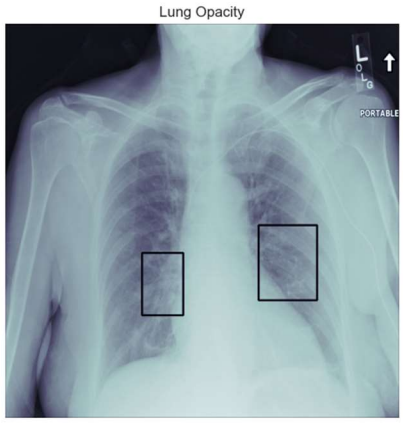
For better display of dicom (with .dcm extension), it is better to convert single channel grayscale to 3 channel RGB images.

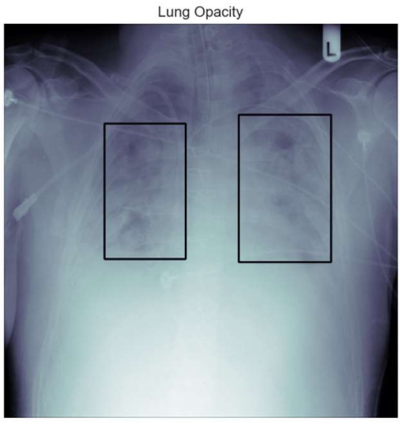
Below snippet is used to convert

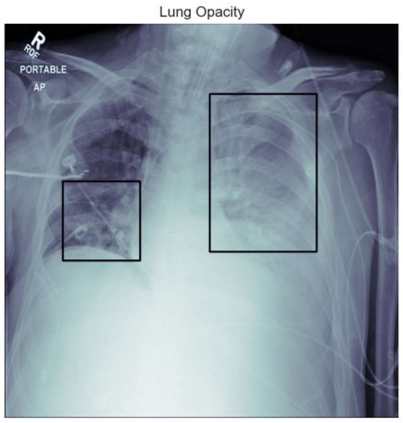
```python
# Importing necessary libraries
import numpy as np
import pydicom

# Function to convert single-channel grayscale DICOM images to 3-channel RGB images
def dcm2rgb(dcm_file):
    """
    This function takes the file path to a DICOM image as input and returns a 3-channel

    Args:
    dcm_file (str): The file path to the DICOM image.

    Returns:
    numpy.ndarray: The resulting 3-channel RGB image.
    """

    if type(dcm_file) == str:
        # Read DICOM file using pydicom
        dcm_data = pydicom.read_file(dcm_file)
    else:
        dcm_data = dcm_file

    # Extract pixel array from DICOM data
    im = dcm_data.pixel_array

    # Convert from single-channel grayscale to 3-channel RGB by stacking the grayscale i
    im = np.stack([im] * 3, axis=2)

    # Resize the image to the desired size (224x224)
    #im = cv2.resize(im, (224, 224))

    # As you need a single channel image, you can convert the RGB image to grayscale
    im = cv2.cvtColor(im, cv2.COLOR_RGB2GRAY)

    return im
```

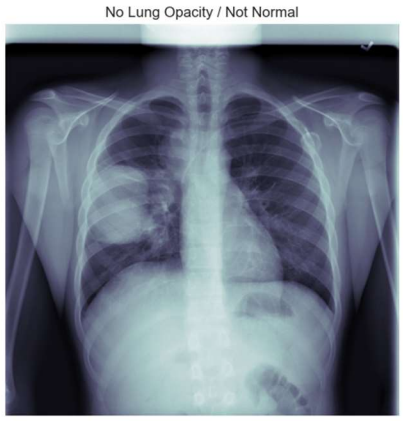
```python
import matplotlib.pyplot as plt
import numpy as np
import cv2

def draw_boxes(image, box, scale_factor):
    # scale coordinates
    x = int(box[0] * scale_factor)
    y = int(box[1] * scale_factor)
    w = int(box[2] * scale_factor)
    h = int(box[3] * scale_factor)

    cv2.rectangle(image, (x, y), (x+w, y+h), color=0, thickness=int(3* scale_factor+0.5)

    return image
# get some sample images from each class
sample_patients = class_info.groupby('class')['patientId'] \
                            .apply(lambda x: list(np.random.choice(x, 3, replace=False))
                            .to_dict()

# define scale factor for DICOM Images
scale_factor = 1024 / 1024

# initialize subplots in a grid 3x3
fig, ax = plt.subplots(3, 3, figsize=(20,20))

for i, cls in enumerate(['Lung Opacity', 'No Lung Opacity / Not Normal', 'Normal']):
    for j in range(3):
        patientId = sample_patients[cls][j]
        dcm_file = f'{train_images_path}/{patientId}.dcm'
        im = dcm2rgb(dcm_file)

        if cls == 'Lung Opacity':
            boxes = train_anno_dict.get(patientId)
            if boxes is not None:
                for box in boxes:
                    im = draw_boxes(im, box, scale_factor)

        ax[i, j].imshow(im, cmap=plt.cm.bone)
        ax[i, j].axis('off')
        ax[i, j].set_title(cls)

plt.show()
```

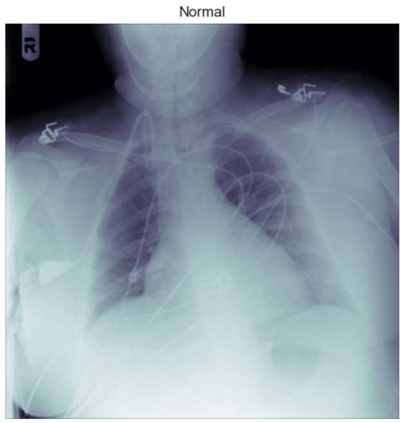
```
import matplotlib.pyplot as plt
import numpy as np
import cv2

def draw_boxes(image, box, scale_factor):
    # scale coordinates
    x = int(box[0] * scale_factor)
    y = int(box[1] * scale_factor)
    w = int(box[2] * scale_factor)
    h = int(box[3] * scale_factor)

    cv2.rectangle(image, (x, y), (x+w, y+h), color=0, thickness=int(3* scale_factor+0.5))

    return image
# get some sample images from each class
sample_patients = class_info.groupby('class')['patientId'] \
                            .apply(lambda x: list(np.random.choice(x, 3, replace=False)))
                            .to_dict()

# define scale factor for DICOM Images
scale_factor = 1024 / 1024

# initialize subplots in a grid 3x3
fig, ax = plt.subplots(3, 3, figsize=(20,20))

for i, cls in enumerate(['Lung Opacity', 'No Lung Opacity / Not Normal', 'Normal']):
    for j in range(3):
        patientId = sample_patients[cls][j]
        dcm_file = f'{train_images_path}/{patientId}.dcm'
        im = dcm2rgb(dcm_file)

        if cls == 'Lung Opacity':
            boxes = train_anno_dict.get(patientId)
            if boxes is not None:
                for box in boxes:
                    im = draw_boxes(im, box, scale_factor)

        ax[i, j].imshow(im, cmap=plt.cm.bone)
        ax[i, j].axis('off')
        ax[i, j].set_title(cls)

plt.show()
```











Display of images with png extension

As mentioned earlier, large sized dicom images are resulting to frequent kernel crash. It is agreed that converting these images to smaller size images with .png extension. Checking whether the images looks similar after conversion.

Below code is used to make this happen

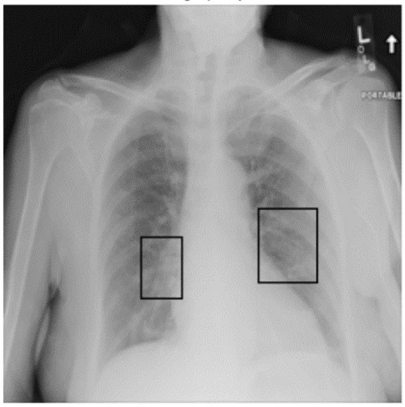
```
import os

def load_image(patientId, metadata_df):
    # Get the folder name for the given patientId
    folder_name = metadata_df.loc[metadata_df['patientId'] == patientId, 'folder'].value

    # Create the full path to the image
    image_path = os.path.join(folder_name, patientId + '.png')

    # Load the image (you'll need an appropriate library like PIL or matplotlib to do th
    # For example, using PIL:
    from PIL import Image
    img = Image.open(image_path)
    img = np.array(img)
    return img
```

Step 5: Display PNG images with bounding box. [5 points]

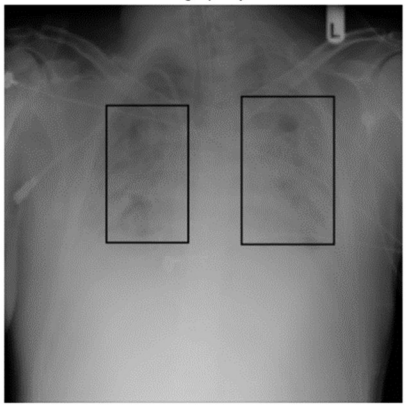
```
# define scale factor for DICOM Images
scale_factor = image_size / 1024

# initialize subplots in a grid 3x3
fig, ax = plt.subplots(3, 3, figsize=(20,20))

for i, cls in enumerate(['Lung Opacity', 'No Lung Opacity / Not Normal', 'Normal']):
    for j in range(3):
        patientId = sample_patients[cls][j]

        im = load_image(patientId, metadata_df)

        if cls == 'Lung Opacity':
            boxes = train_anno_dict.get(patientId)
            if boxes is not None:
                for box in boxes:
                    im = draw_boxes(im, box, scale_factor)

        ax[i, j].imshow(im, cmap=plt.cm.bone)
        ax[i, j].axis('off')
        ax[i, j].set_title(cls)

plt.show()
```

Displaying the images with .png extension

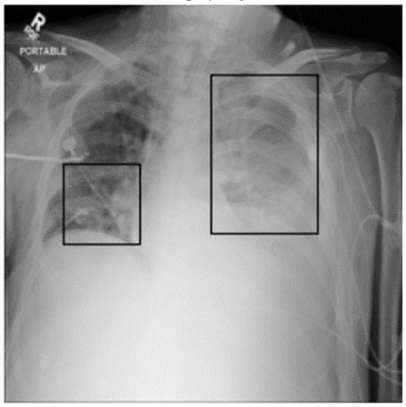
Lung Opacity



Lung Opacity

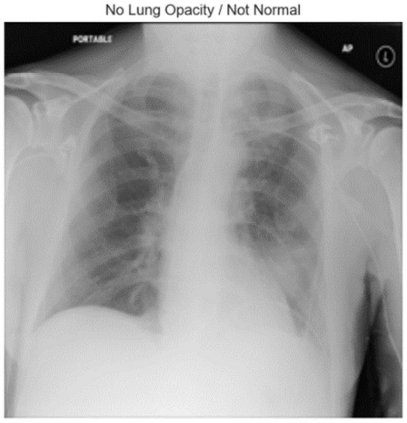


Lung Opacity

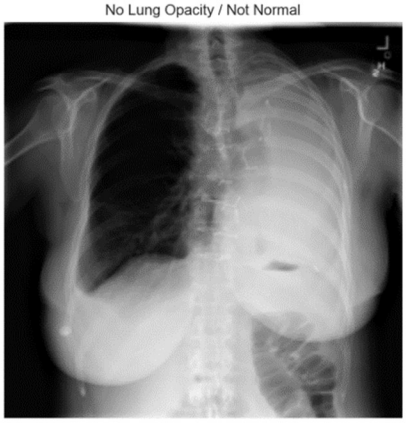


No Lung Opacity / Not Normal

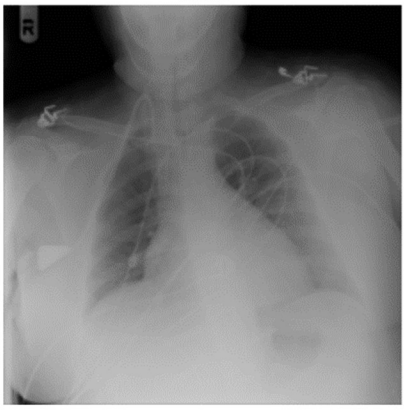
No Lung Opacity / Not Normal



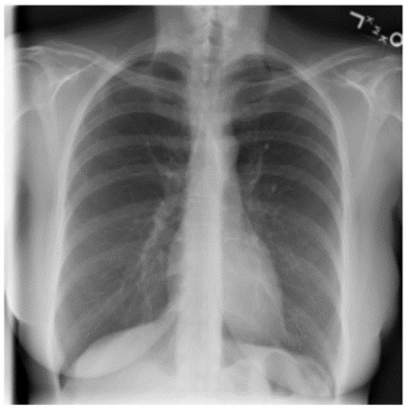
No Lung Opacity / Not Normal



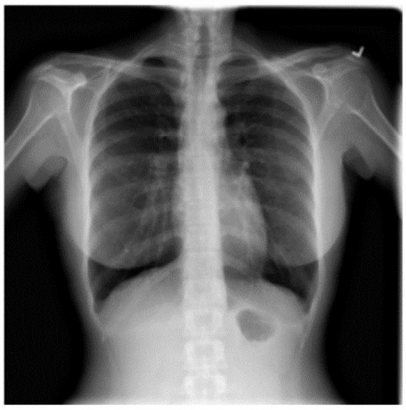
Normal



Normal



Normal



Building the CNN Model

Attempt 1- Using DICOM images

Now it is time to build the CNN model. Here, first attempt is to try with limited (15000) dicom images. For this required libraries are imported and started building the model. During this images are used for both training and evaluation

This method involves reading the DICOM training images in memory and convert the numpy array and use that for training. It was observed that memory requirement is high and the kernel usually dies. Hence only limited number of (10000) images are loaded. The decision was made to go for binary classification based on Target column of stage_2_train_labels.csv

```
# Usual Imports
import tensorflow as tf
import numpy as np
import pydicom
import os

from skimage.transform import resize
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor

from keras import backend as K
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras import losses, optimizers

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Layer, Convolution2D, Dense, RandomRotation, RandomFlip,
Resizing, Rescaling
from tensorflow.keras.layers import Concatenate, UpSampling2D, Conv2D, Reshape,
GlobalAveragePooling2D, GlobalMaxPooling2D
from tensorflow.keras.layers import Dense, Activation, Flatten, Dropout, MaxPooling2D,
BatchNormalization
from tensorflow.keras.layers import Input, ReLU, AveragePooling2D, Flatten
from tensorflow.keras.layers import SeparableConv2D, MaxPool2D
from tensorflow.keras.layers import Dropout, RandomRotation

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau, LearningRateScheduler

from tensorflow.keras.regularizers import l2,  l1  # L2 regularization
from tensorflow.keras.metrics import Recall

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Read the Metadat prepared from the provided csv files
metadatfilename = 'Metadata'+id_str+'.csv'
metadata_df = pd.read_csv(metadatfilename)
```

Below are the code snippets defines parameters for building the model

```
# Following code sets various parameters for model building.
# This gives us flexibility of trying binary or multiclass classification with minimal changes to
the code
# Only setting n_classes and c_mode gives us the flexibility of building the models witth
appropriate parameters

classes2use = 'label'+str(n_classes)
```

```
c_mode='binary'

# Select model parameeters based on the selected training requirements

if n_classes == 2:
    if c_mode == 'binary':
        lbl = lbl2
        cls_mode = 'binary'
        actvn = 'sigmoid'
        lss = 'binary_crossentropy'
        num_classes  = 1
    else:
        lbl = lbl2
        cls_mode = 'categorical'
        actvn = 'softmax'
        lss = 'categorical_crossentropy'
        num_classes  = 2
else:
    lbl = lbl3
    cls_mode = 'categorical'
    actvn = 'softmax'
    lss = 'categorical_crossentropy'
    num_classes  = 3

# code below can also run for greyscale or color images
n_channels = len(set_color)

# Rescale annotation from 1024 to current image size
def rescale_annotations(annotations, scale_factor):
    rescaled_annotations = []
    for annotation in annotations:
        # Assuming annotation is a bounding box represented as (x, y, width, height)
        x, y, width, height = annotation
        # Rescale the coordinates using the scale factor
        new_x = x * scale_factor
        new_y = y * scale_factor
        new_width = width * scale_factor
        new_height = height * scale_factor
        rescaled_annotations.append((int(new_x), int(new_y), int(new_width), int(new_height)))
    return rescaled_annotations

# Extract the image data from the dicom images and convert it to numpy array, resize it store it
in imgs
import cv2
ALPHA = 1
IMAGE_HEIGHT = image_size
IMAGE_WIDTH = image_size
scale_factor = image_size/ORIG_SIZE

X = []
annot = []
masks = np.zeros((int(train_metadata.shape[0]), IMAGE_HEIGHT, IMAGE_WIDTH, 1))  # Add an
additional dimension for channel

for index, patid in enumerate(tqdm(train_metadata['patientId'])):
    # Read Images and populate them and load them in memory
    im = pydicom.dcmread(os.path.join(train_images_path, f"{patid}.dcm"))
    im = im.pixel_array
    im = np.stack((im,) * n_channels, -1)

    im = np.array(im).astype(np.uint8)
    im = cv2.resize(im, (image_size, image_size), interpolation = cv2.INTER_LINEAR)

    # Standardize the image (for both grayscale and RGB)
    im = (im - np.min(im)) / (np.max(im) - np.min(im))

    X.append(im)

    # Read & Resize boxes and populate them and load them in memory
    boxes = train_anno_dict.get(patid, [])
    ann = rescale_annotations(boxes, scale_factor)
```

```
    annot.append(ann)
    for box in ann:
        x1 = box[0]
        y1 = box[1]
        x2 = box[2] + x1
        y2 = box[3] + y1
        masks[index][y1:y2, x1:x2, 0] = 1  # Add the mask to the corresponding channel
```

Changing the data into train and test images.

```
# Create X and Y for training
#X = X.reshape((X.shape[0], X.shape[1], X.shape[2], n_channels))
X = np.array(X)
print(f"Train shape {X.shape}")
from tensorflow.keras.utils import to_categorical
if c_mode == 'binary':
    Y = train_metadata[classes2use]
else:
    Y = to_categorical(train_metadata[classes2use])

print("Target shape ", Y.shape)
print("Mask shape ", masks.shape)

from sklearn.model_selection import train_test_split

# Split the data into temp training and test sets
X_train, X_test, Y_train, Y_test, Ym_train, Ym_test = train_test_split(
    X, Y, masks, test_size=0.2, random_state=42, stratify=Y, shuffle=True)

# Split the temporary training set into final train and validation sets
X_train, X_val, Y_train, Y_val, Ym_train, Ym_val = train_test_split(
    X_train, Y_train, Ym_train, test_size=0.25, random_state=42, stratify=Y_train,
shuffle=True)
```

Building the model

1. CNN for classification with Recall

When using CNN (Convolutional Neural Network) for classification with recalls, it is important to consider the specific requirements and goals of the classification task. Recall is a metric that measures the ability of a classification model to identify all relevant instances within a dataset.

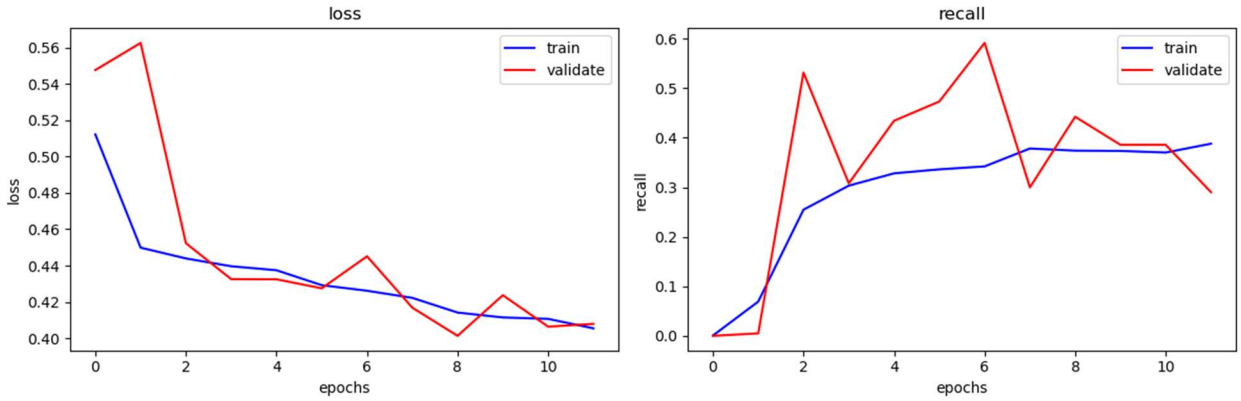
In the context of CNN, the recall can be optimized by adjusting various parameters and techniques. One important consideration is the choice of network architecture, as different architectures can impact recall performance. Additionally, the use of data augmentation techniques can help increase the diversity and size of the training data, which can lead to improved recall rates.

Furthermore, optimizing the loss function can be beneficial for maximizing recall. For instance, using a loss function like binary cross-entropy.

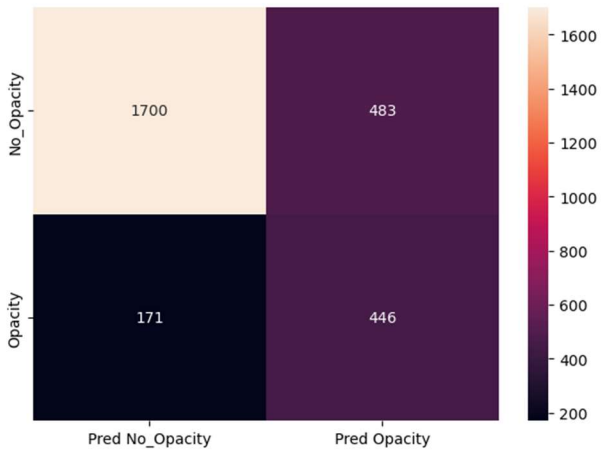
```
img_dims = image_size
inputs = Input(shape=(img_dims, img_dims, n_channels))

# First conv block
x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Second conv block
```

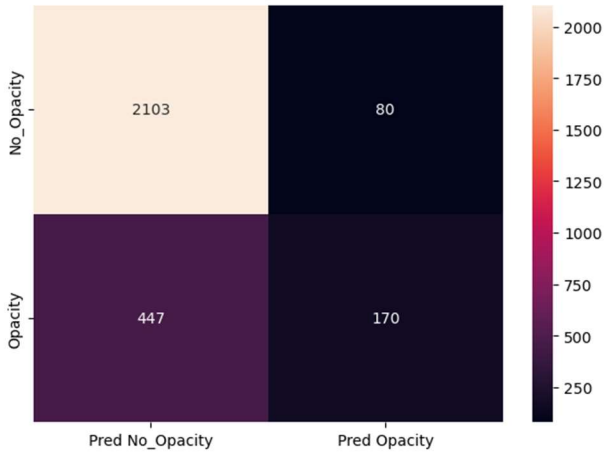
```
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Third conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)

# Fourth conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.3)(x)

# Fifth conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.4)(x)

# Sixth conv block
x = SeparableConv2D(filters=512, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=512, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.5)(x)

# FC layer
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(rate=0.7)(x)
x = Dense(units=256, activation='relu')(x)
x = Dropout(rate=0.5)(x)
x = Dense(units=128, activation='relu')(x)
x = Dropout(rate=0.3)(x)

# Output layer
output = Dense(units=num_classes, activation=actvn)(x)

# Creating model and compiling
model = tf.keras.Model(inputs=inputs, outputs=output)
mmodel.summary()

Desc = 'CNN for classification - with recall'
Saved_Model = 'CNN- DICOM 224r.h5'
start_time = time()
```

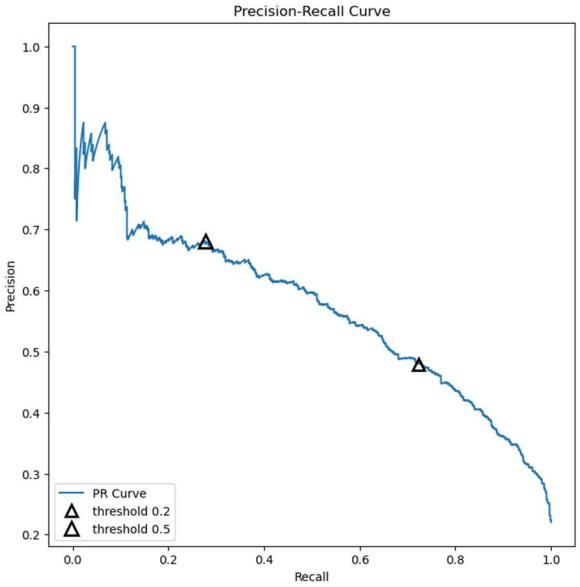
```
# Callbacks
checkpoint = ModelCheckpoint(Saved_Model, save_best_only=True, save_weights_only=True,
verbose =1)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=3, verbose=2,
mode='min')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=10, mode='min',
restore_best_weights = True)

# Add data augmentation
datagen = ImageDataGenerator(rotation_range=0.05, zoom_range=0.15,
width_shift_range=0.15, height_shift_range=0.15)
datagen.fit(X_train)

history = model.fit(datagen.flow(X_train, Y_train, batch_size=batch_size),
                    epochs = EPOCHS_SET,
                    validation_data = (X_val, Y_val), callbacks=[checkpoint, lr_reduce,
early_stop])
end_time = time()odel.compile(optimizer=Adam(learning_rate=0.0001), loss=lss,
metrics=[Recall()])
```

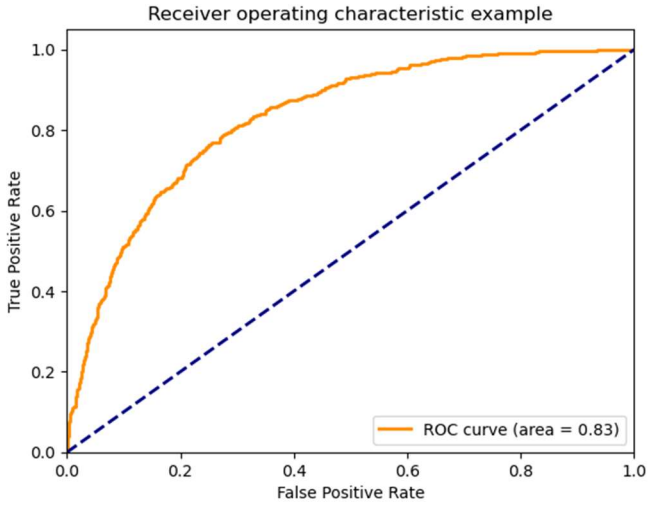



```
88/88 [==============================] - 4s 42ms/step
Threshold = 0.2:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.91      0.78      0.84      2183
     Opacity       0.48      0.72      0.58       617

    accuracy                           0.77      2800
   macro avg       0.69      0.75      0.71      2800
weighted avg       0.81      0.77      0.78      2800

Threshold = 0.5:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.82      0.96      0.89      2183
     Opacity       0.68      0.28      0.39       617

    accuracy                           0.81      2800
   macro avg       0.75      0.62      0.64      2800
weighted avg       0.79      0.81      0.78      2800
```










```
results_df
```

	Description	Threshold	Accuracy	Precision	Recall	F1 Score	Model	Execution Time
0	CNN for classification - with recall	0.2	0.766429	0.480086	0.722853	0.576973	CNN- DICOM 224r.h5	560.580348
1	CNN for classification - with recall	0.5	0.811786	0.680000	0.275527	0.392157	CNN- DICOM 224r.h5	560.580348

2. CNN for classification- with Recall & Accuracy

When using CNNs (Convolutional Neural Networks) for classification tasks, it is important to consider both recall and accuracy as evaluation metrics.

Accuracy measures how often the model correctly predicts the class label, while recall measures the ability of the model to correctly identify the positive instances.

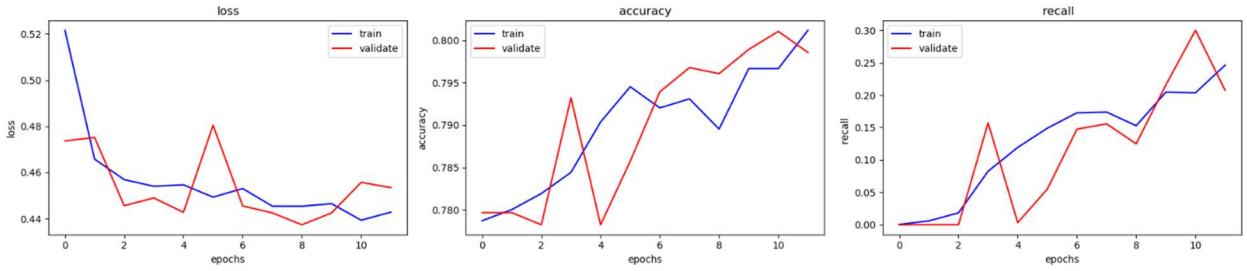
In a classification problem where the class distribution is imbalanced, accuracy may not provide an accurate measure of the model's performance. This is because the model can achieve high accuracy by simply predicting the majority class. In such cases, recall becomes more important as it focuses on correctly identifying the positive instances, which may be of higher importance.

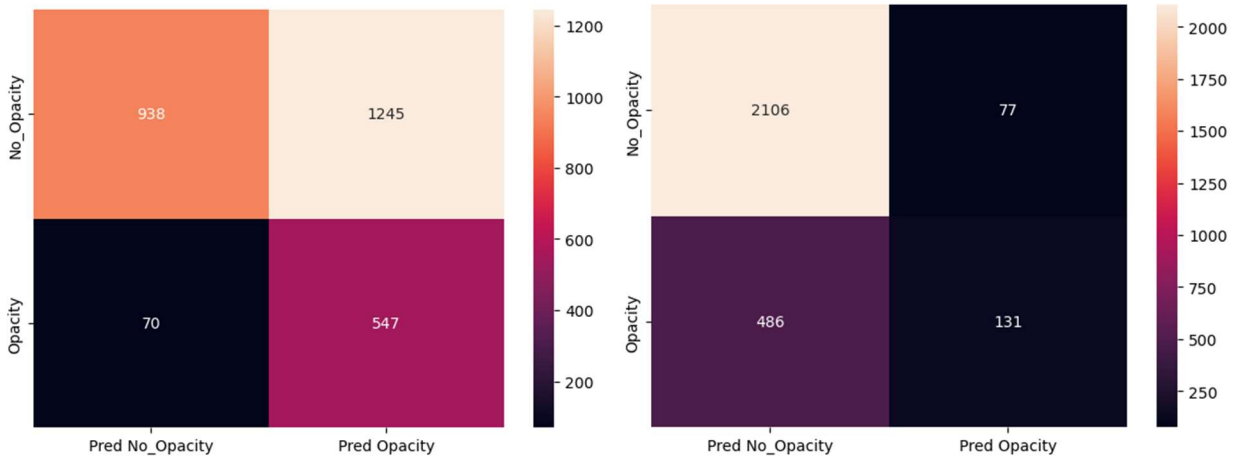
To optimize both recall and accuracy, one can use techniques such as adjusting the classification threshold

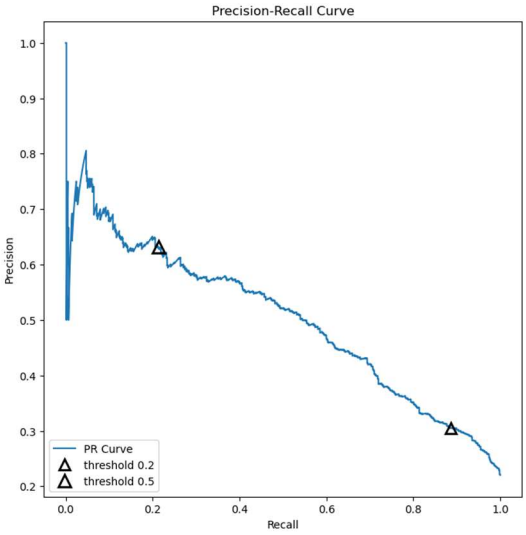
```
ADJUSTED_IMAGE_SIZE = image_size
input_shape = (ADJUSTED_IMAGE_SIZE, ADJUSTED_IMAGE_SIZE, n_channels)

model = Sequential()
#model.add(RandomRotation(factor=0.15))
#model.add(Rescaling(1./255))
model.add(Conv2D(32, (3, 3), input_shape=input_shape)) # (3, 3) - conv kernel

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3)))

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(256, (3, 3)))

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation(actvn))

# Add data augmentation
datagen = ImageDataGenerator(rotation_range=2, zoom_range=0.15,
width_shift_range=0.15, height_shift_range=0.15)
datagen.fit(X_train)

model.compile(loss=lss,
              optimizer=Adam(learning_rate=0.001, decay=0.001/100),
              metrics=['accuracy', Recall()] # ,f1_m'accuracy' for i in
range(num_classes), decay=0.004
             )
model.summary()

Desc = 'CNN for classification - with recall & accuracy'
Saved_Model = 'CNN- DICOM 224-ra.h5'
start_time = time()

# Callbacks
```

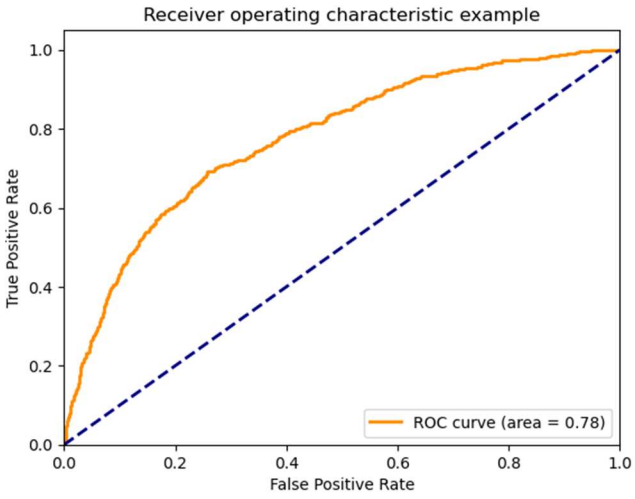
```
history = model.fit(datagen.flow(X_train, Y_train, batch_size=batch_size),
                    epochs = EPOCHS_SET,
                    validation_data = (X_val, Y_val),
                      callbacks=[
                          ModelCheckpoint(Saved_Model, save_best_only=True, verbose = 1,
save_weights_only=True),
                          EarlyStopping(monitor = "val_loss", patience = 10,
restore_best_weights = True),
                          ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
mode='min')
                       ]
                     )
end_time = time()
```

```
    Classification Report

    88/88 [==============================] - 1s 7ms/step
    Threshold = 0.2:
    Classification Report
                  precision    recall  f1-score   support

      No_Opacity       0.93      0.43      0.59      2183
         Opacity       0.31      0.89      0.45       617

        accuracy                           0.53      2800
       macro avg       0.62      0.66      0.52      2800
    weighted avg       0.79      0.53      0.56      2800


    Threshold = 0.5:
    Classification Report
                  precision    recall  f1-score   support

      No_Opacity       0.81      0.96      0.88      2183
         Opacity       0.63      0.21      0.32       617

        accuracy                           0.80      2800
       macro avg       0.72      0.59      0.60      2800
    weighted avg       0.77      0.80      0.76      2800
```









2	CNN for classification - with recall & accuracy	0.2	0.530357	0.305246	0.886548	0.454130	CNN-DICOM 224-ra.h5	495.309162
3	CNN for classification - with recall & accuracy	0.5	0.798929	0.629808	0.212318	0.317576	CNN-DICOM 224-ra.h5	495.309162

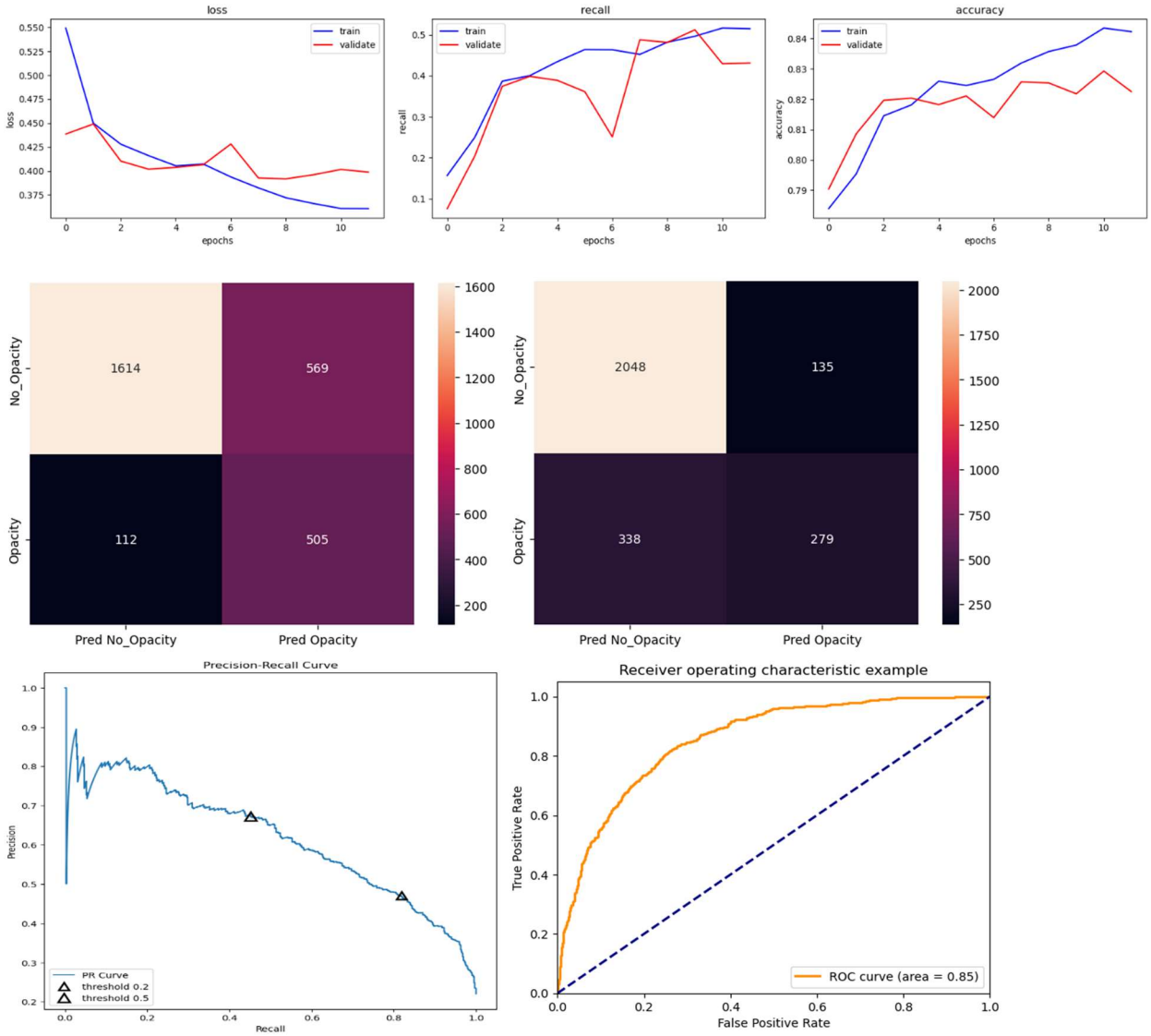
3. Transfer Learning for classification- VGG16 with Recall & Accuracy

Transfer Learning is a powerful technique that leverages knowledge learned from one task to improve performance on a related task. In the context of classification, transfer learning allows us to apply pre-trained models on one dataset to another dataset with similar characteristics.

By using transfer learning, we can benefit from the knowledge and features learned by deep neural networks on large datasets like ImageNet. Instead of training a model from scratch on a small dataset, we can fine-tune a pre-trained model to the new dataset. This approach reduces the need for large amounts of labeled data and improves the generalization capability of the model.

To implement transfer learning for classification, we typically remove the last few layers of the pre-trained model

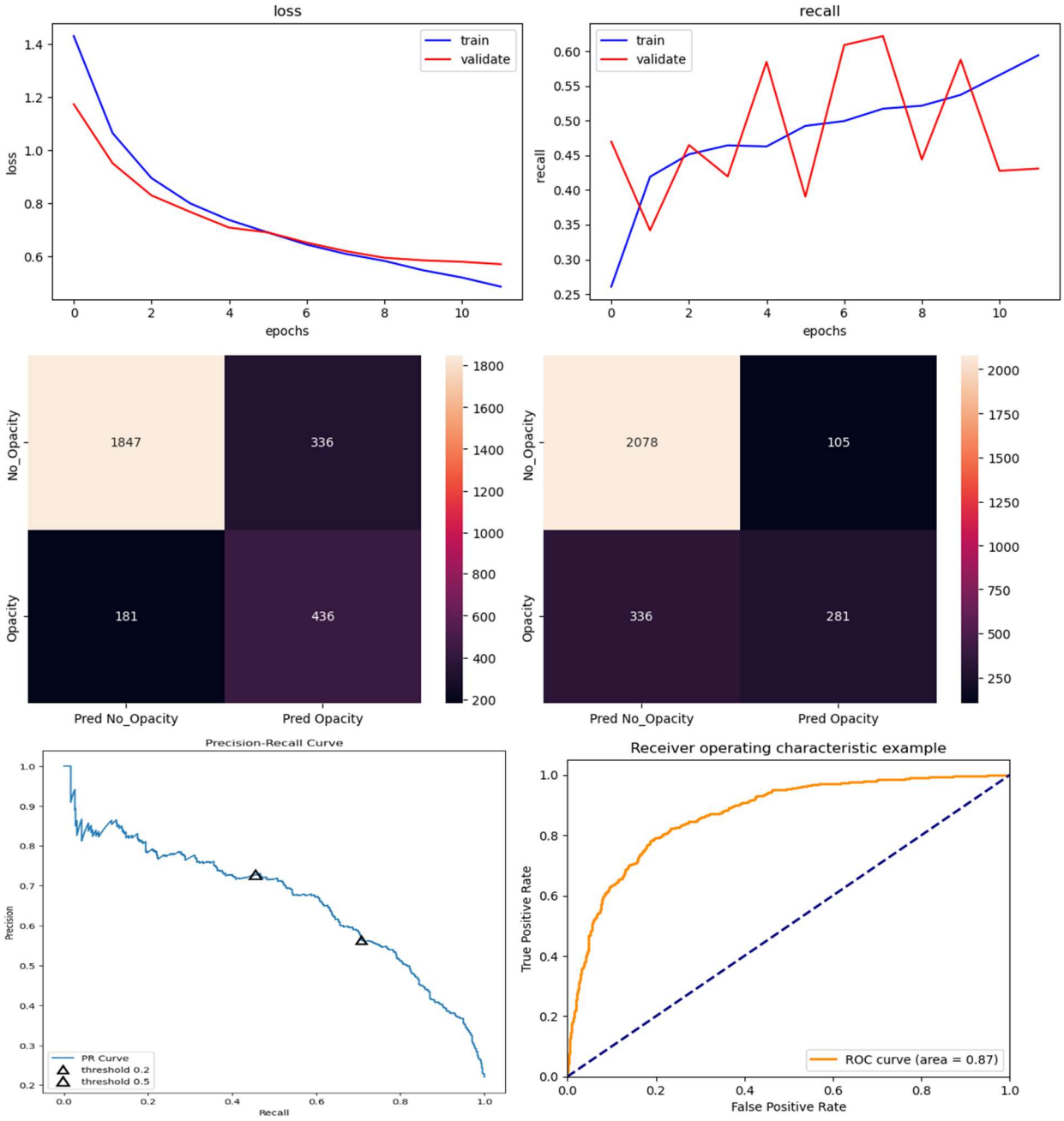
```
ADJUSTED_IMAGE_SIZE = image_size
input_shape = (ADJUSTED_IMAGE_SIZE, ADJUSTED_IMAGE_SIZE, n_channels)
base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False)
for layer in base_model.layers:
    layer.trainable = False
# Set last 3 layers to be trainable
for layer in base_model.layers[-3:]:
    layer.trainable = True

inputs = tf.keras.Input(shape=input_shape, name='inputLayer')
#x = augment(inputs)
pretrain_out = base_model(inputs, training=False)
x = layers.Flatten()(pretrain_out)
x = layers.Dense(64, activation='relu', kernel_regularizer=l2(0.01))(x) # Add L2
regularization
x = layers.Dropout(0.5)(x)  # Add dropout
x = layers.Dense(num_classes, name='outputLayer')(x)
outputs = layers.Activation(activation=actvn, dtype=tf.float32,
name='activationLayer')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005,
decay=0.001/100),
              loss=lss,
              metrics=[Recall(), 'accuracy'])

# Add data augmentation
datagen = ImageDataGenerator(rotation_range=2, zoom_range=0.15,
width_shift_range=0.15, height_shift_range=0.15)
datagen.fit(X_train)

model.summary()

Desc = 'VGG16 with recall & accuracy'
Saved_Model = 'VGG16- DICOM 224 ra.h5'

88/88 [==============================] - 3s 35ms/step
Threshold = 0.2:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.94      0.74      0.83      2183
     Opacity       0.47      0.82      0.60       617

    accuracy                           0.76      2800
   macro avg       0.70      0.78      0.71      2800
weighted avg       0.83      0.76      0.78      2800

Threshold = 0.5:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.86      0.94      0.90      2183
     Opacity       0.67      0.45      0.54       617

    accuracy                           0.83      2800
   macro avg       0.77      0.70      0.72      2800
weighted avg       0.82      0.83      0.82      2800
```



4. Transfer Learning for classification- VGG16 with recall

```
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
ADJUSTED_IMAGE_SIZE = image_size
input_shape = (ADJUSTED_IMAGE_SIZE, ADJUSTED_IMAGE_SIZE, n_channels)
base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False)

base_model.trainable = True
for layer in base_model.layers:
    if isinstance(layer, BatchNormalization): # set BatchNorm layers as not trainable
        layer.trainable = False

inputs = tf.keras.Input(shape=input_shape, name='inputLayer')
#x = augment(inputs)
pretrain_out = base_model(inputs, training=False)
x = layers.Flatten()(pretrain_out)
```

```
x = layers.Dense(64, activation='relu', kernel_regularizer=l2(0.01))(x) # Add L2
regularization
x = layers.Dropout(0.5)(x)  # Add dropout
x = layers.Dense(num_classes, name='outputLayer')(x)

outputs = layers.Activation(activation=actvn, dtype=tf.float32,
name='activationLayer')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
              loss=lss,
              metrics=[Recall()])

# Add data augmentation
datagen = ImageDataGenerator(rotation_range=2, zoom_range=0.15,
width_shift_range=0.15, height_shift_range=0.15)#
datagen.fit(X_train)

model.summary()

Desc = 'VGG16 with recall'
Saved_Model = 'VGG16- DICOM 224 r.h5'
start_time = time()

history = model.fit(datagen.flow(X_train, Y_train, batch_size=16),
                    epochs = EPOCHS_SET,
                    validation_data=(X_val, Y_val),
                    callbacks=[
                        ModelCheckpoint(Saved_Model, save_best_only=True, verbose = 1,
save_weights_only=True),
                        EarlyStopping(monitor="val_loss", patience=7,
restore_best_weights=True, mode='min'),
                        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2,
mode='min')
                    ])
end_time = time()

88/88 [==============================] - 3s 35ms/step
Threshold = 0.2:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.91      0.85      0.88      2183
     Opacity       0.56      0.71      0.63       617

    accuracy                           0.82      2800
   macro avg       0.74      0.78      0.75      2800
weighted avg       0.83      0.82      0.82      2800

Threshold = 0.5:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.86      0.95      0.90      2183
     Opacity       0.73      0.46      0.56       617

    accuracy                           0.84      2800
   macro avg       0.79      0.70      0.73      2800
weighted avg       0.83      0.84      0.83      2800
```

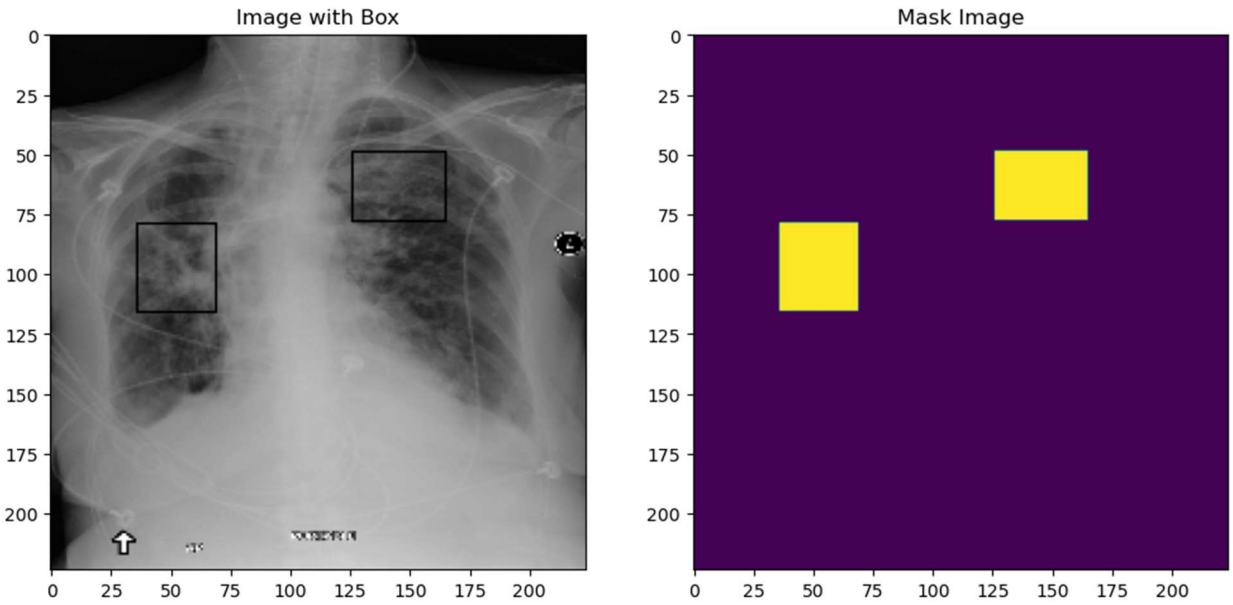


Design, train and test RCNN & its hybrids based object detection models to impose the bounding box or mask over the area of interest.

5. Object Detection : Mask Detection Using Mask R-CNN

Object detection is a computer vision task that involves identifying and localizing objects in an image or a video. It is widely used in various applications such as autonomous driving, surveillance, and object recognition. Mask detection, on the other hand, is a specific type of object detection that focuses on identifying whether a person is wearing a mask or not.

One popular approach for object detection is called Mask R-CNN. This method combines the use of convolutional neural networks (CNNs) for object detection with the addition of a mask prediction network. The mask prediction network predicts a binary mask for each detected object, indicating the pixel-wise segmentation of the object.



Model definition

A convolutional neural network (CNN) that includes elements of a residual network (ResNet) is a powerful deep learning architecture. CNNs are widely used for image classification tasks, thanks to their ability to capture local spatial features using convolutional layers. On the other hand, ResNets address the challenge of training very deep neural networks by introducing skip connections that enable the network to learn residual functions.

By combining these two techniques, a CNN with ResNet elements can benefit from both spatial feature extraction and the ability to train deep networks effectively. The skip connections in ResNet allow the network to leverage information from earlier layers, ensuring that the gradients flow more smoothly during backpropagation.

The code contains downsample layers and residual blocks, making it more similar to a variant of the ResNet model rather than the Mask R-CNN. Specifically, it appears to have a ResNet-style architecture with additional batch normalization and leaky ReLU activation functions, as well as a final layer that applies a sigmoid activation function to a 1x1 convolution, followed by upsampling.

```
# create 1 downsample layer, each containing 4 layers in it
def create_downsample(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

# creates 1 residual layer, each containing 6 layers in it.
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])

# Model creator
# Depth = number of layers in the model
def create_network(input_size, channels, n_blocks=2, depth=4):
    # input layers - 2 layer
    inputs = keras.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, n_channels))
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(inputs)
    print("After Conv2D:", x.shape)

    # residual blocks (4*4 downsample + 4*2*6 resblock = 64 layers)
    for d in range(depth):
        channels = channels * 2
        x = create_downsample(channels, x)
        print(f"After downsample {d + 1}:", x.shape)
        for b in range(n_blocks):
            x = create_resblock(channels, x)
            print(f"After resblock {d + 1}.{b + 1}:", x.shape)

    # output - 4 layers
    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
```

In machine learning, IOU (Intersection over Union) and Jaccard loss are both metrics commonly used in evaluating the performance of object detection and segmentation models.

IOU is a measure of the overlap between the predicted and ground truth bounding boxes or masks. It calculates the ratio of the area of intersection between the predicted and ground truth regions to the area of union between them. IOU is used to quantify how well the model's predicted regions align with the ground truth regions, with higher values indicating better alignment.

Jaccard loss, on the other hand, is a loss function that is minimized during model training. It is calculated as 1 minus the Jaccard Index, which is the same.

```
# define iou or jaccard loss function
def iou_loss(y_true, y_pred):
    y_true = tf.reshape(y_true, [-1])
    y_pred = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true * y_pred)
    score = (intersection + 1.) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) -
intersection + 1.)
    return 1 - score

# combine bce loss and iou loss
def iou_bce_loss(y_true, y_pred):
    return 0.25 * keras.losses.binary_crossentropy(y_true, y_pred) + 0.75 *
iou_loss(y_true, y_pred)
```

```
def mean_iou(y_true, y_pred):
    y_pred = tf.round(y_pred)  # Ensure predictions are binary (0 or 1)

    intersection = tf.reduce_sum(y_true * y_pred, axis=[1, 2])
    union = tf.reduce_sum(y_true + y_pred, axis=[1, 2]) - intersection

    iou = (intersection + tf.keras.backend.epsilon()) / (union +
tf.keras.backend.epsilon())

    return tf.reduce_mean(iou)

from tensorflow import keras
# create network and compiler
model = create_network(input_size=(IMAGE_HEIGHT, IMAGE_WIDTH, n_channels),
channels=32, n_blocks=2, depth=4)
model.compile(optimizer='adam',
              loss=iou_bce_loss,
              metrics=[Recall(), 'accuracy', mean_iou])
model.summary()
```

The cosine learning rate is a type of learning rate schedule commonly used in deep learning. Unlike traditional learning rate schedules, which typically decrease the learning rate linearly or exponentially, the cosine learning rate follows a cosine-shaped curve.

The main advantage of using a cosine learning rate is that it allows for a smooth transition between high and low learning rates during training. This can help improve the convergence of the model and prevent it from getting stuck in local minima.

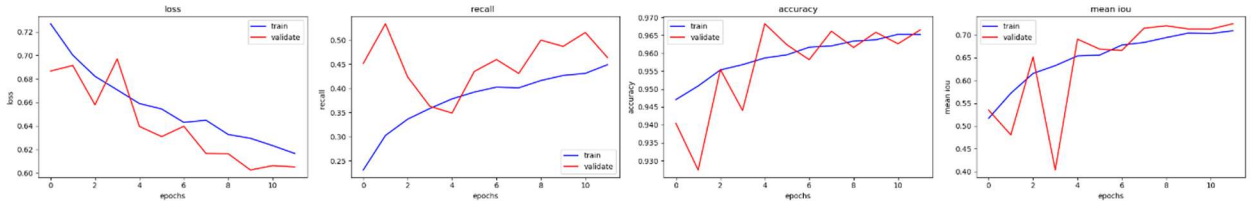
The cosine learning rate is computed using the formula:

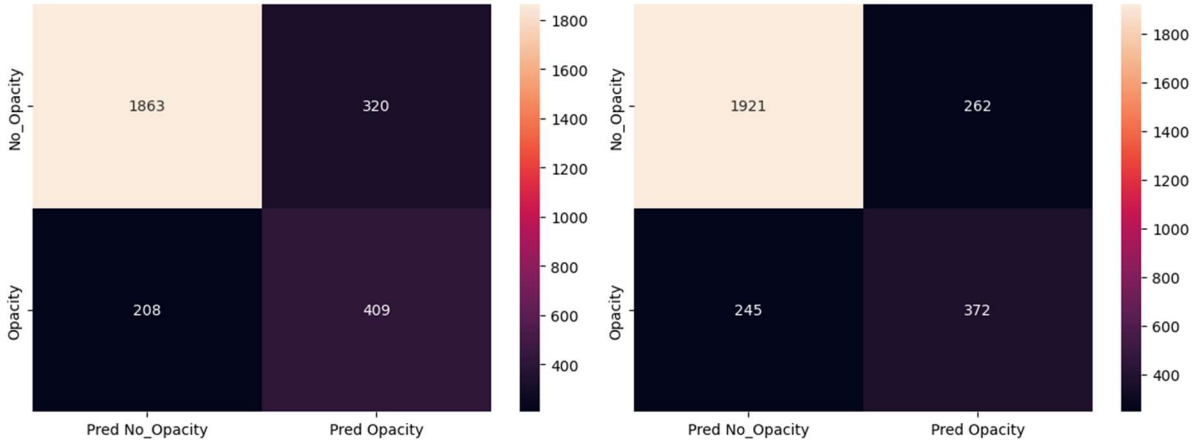
learning_rate = initial_learning_rate * 0.5 * (1 + cos(epoch / max_epochs * pi))

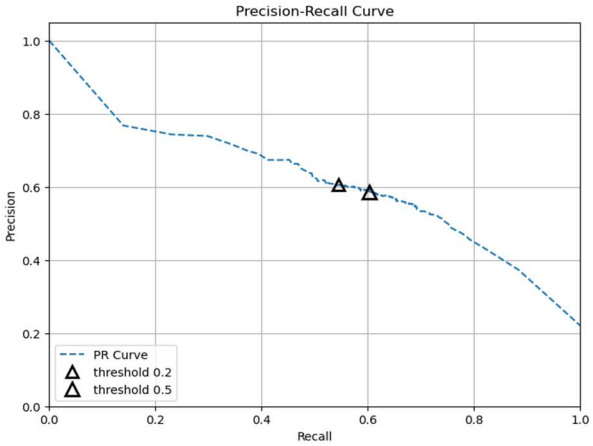
where epoch is the current epoch and max_epochs is the total number of epochs.

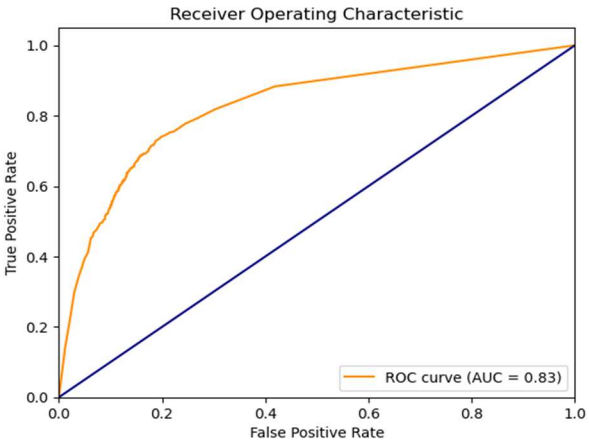
```
# cosine learning rate annealing
# changes learning rate based on the number of epocs passed


def cosine_annealing(x):
    lr = 0.0007
    epochs = EPOCHS_SET
    return lr* (np.cos(np.pi*x/epochs)+1.) /2

learning_rate = LearningRateScheduler(cosine_annealing)

Desc = 'Object Detection- Mask R-CNN'
Saved_Model = 'Mask R-CNN- DICOM 224.h5'

# Saves training time once the best model is achieved.
checkpoint = ModelCheckpoint(Saved_Model, verbose = 1, save_best_only = True,
save_weights_only=True)

# Keep monitoring val_loss to see if there is any improvement.
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.0005, patience=15,
                                                    restore_best_weights=True,
verbose=1, mode='auto')

# Add data augmentation
```

```
datagen = ImageDataGenerator(rotation_range=2, zoom_range=0.15,
width_shift_range=0.15, height_shift_range=0.15)#
datagen.fit(X_train)


start_time = time()
#training the model
history = model.fit(datagen.flow(X_train, Ym_train, batch_size=batch_size),
                    epochs = EPOCHS_SET,
                    validation_data=(X_val, Ym_val),
                    callbacks=[learning_rate, checkpoint, early_stopping])
end_time = time()

88/88 [==============================] - 3s 35ms/step
Threshold: 0.2
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.90      0.85      0.88      2183
     Opacity       0.56      0.66      0.61       617

    accuracy                           0.81      2800
   macro avg       0.73      0.76      0.74      2800
weighted avg       0.82      0.81      0.82      2800

Threshold: 0.5
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.89      0.88      0.88      2183
     Opacity       0.59      0.60      0.59       617

    accuracy                           0.82      2800
   macro avg       0.74      0.74      0.74      2800
weighted avg       0.82      0.82      0.82      2800
```

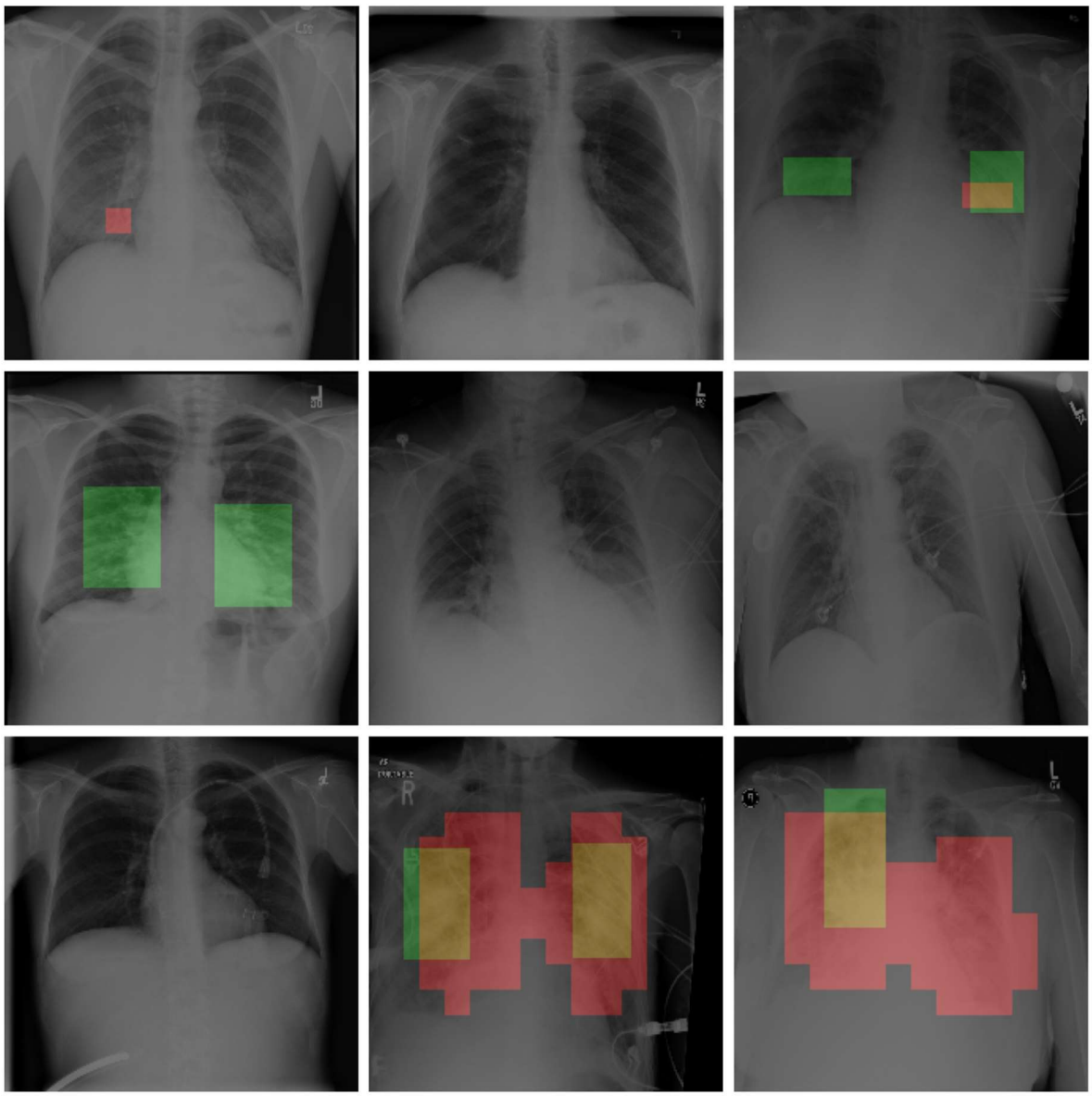





Precision-Recall Curve
Precision
Recall
PR Curve
threshold 0.2
threshold 0.5
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.83)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Results



B) with converted Dicom images in PNG format used for training and evaluation

This method involves converting the DICOM training images to png format and relocating them folders suitable for flow_from_directory method in keras. This way training can be done all images without putting undue stress on memory requirements and make code unworkable. Here again we go for binary classification based on **Target** column of stage_2_train_labels.csv

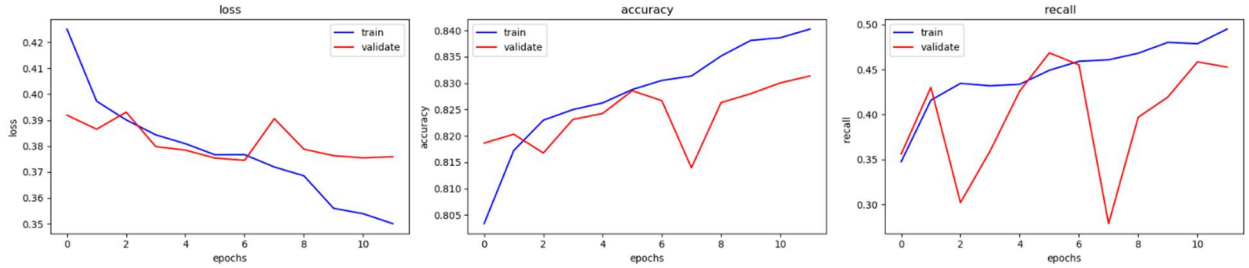
Use of ImageDataGenerator for using all images in PNG format.

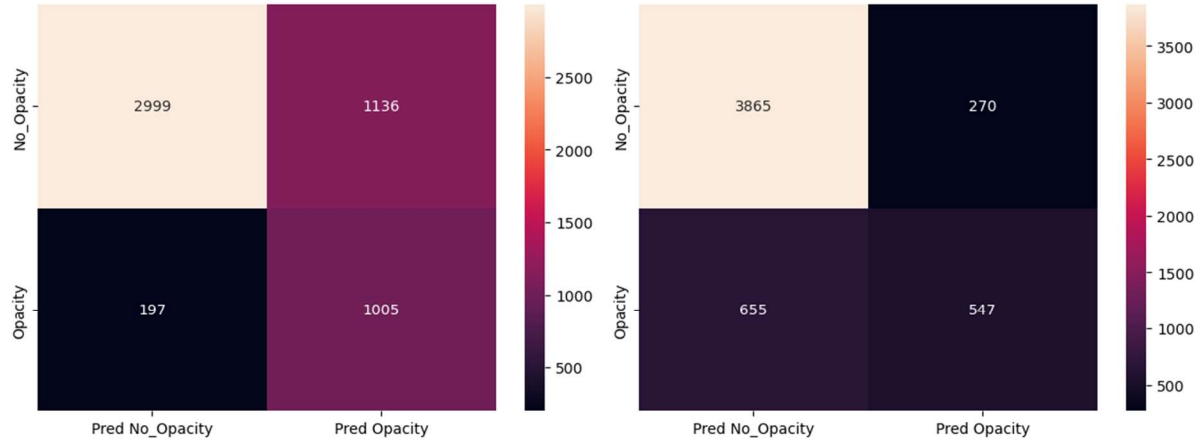
```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen_train = ImageDataGenerator(rescale=1./255, zoom_range = 0.2, vertical_flip
=True, rotation_range=2)#

train_set = datagen_train.flow_from_directory(train_png_images_path,
                                              target_size = (image_size, image_size),
                                              color_mode = color_mode_dict[set_color],
                                              batch_size = batch_size,
                                              class_mode = cls_mode,
                                              classes = lbl,
                                              shuffle = True)

datagen_validation = ImageDataGenerator(rescale=1./255)

validation_set = datagen_validation.flow_from_directory(val_png_images_path,
                                              target_size = (image_size, image_size),
                                              color_mode = color_mode_dict[set_color],
                                              batch_size = batch_size,
                                              class_mode = cls_mode,#'categorical'
                                              classes = lbl,
                                              shuffle = True)
```

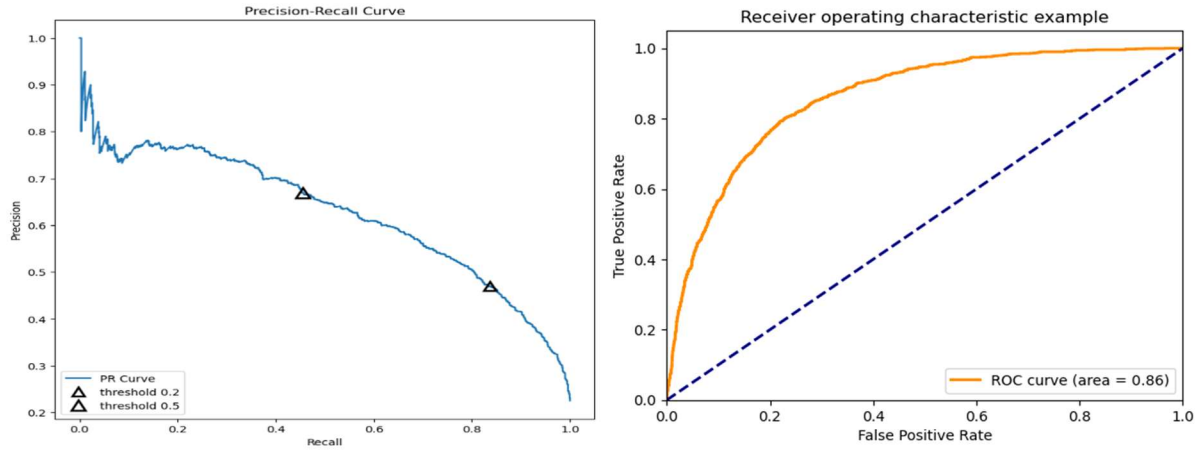
6. VGG16 with PNG files used with flow from directory from Image Data Generator

```
history = model.fit(train_set, validation_data= validation_set,
                    epochs = EPOCHS_SET,
                    callbacks=[
                        EarlyStopping(monitor="val_loss", patience=5,
restore_best_weights=True, mode='min'),
                        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2,
mode='min'),
                        ModelCheckpoint(Saved_Model, verbose = 1, save_best_only =
True, save_weights_only=True)

334/334 [==============================] - 6s 18ms/step
Threshold = 0.2:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.94      0.73      0.82      4135
     Opacity       0.47      0.84      0.60      1202

    accuracy                           0.75      5337
   macro avg       0.70      0.78      0.71      5337
weighted avg       0.83      0.75      0.77      5337
```

```
Threshold = 0.5:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.86      0.93      0.89      4135
     Opacity       0.67      0.46      0.54      1202

    accuracy                           0.83      5337
   macro avg       0.76      0.69      0.72      5337
weighted avg       0.81      0.83      0.81      5337
```





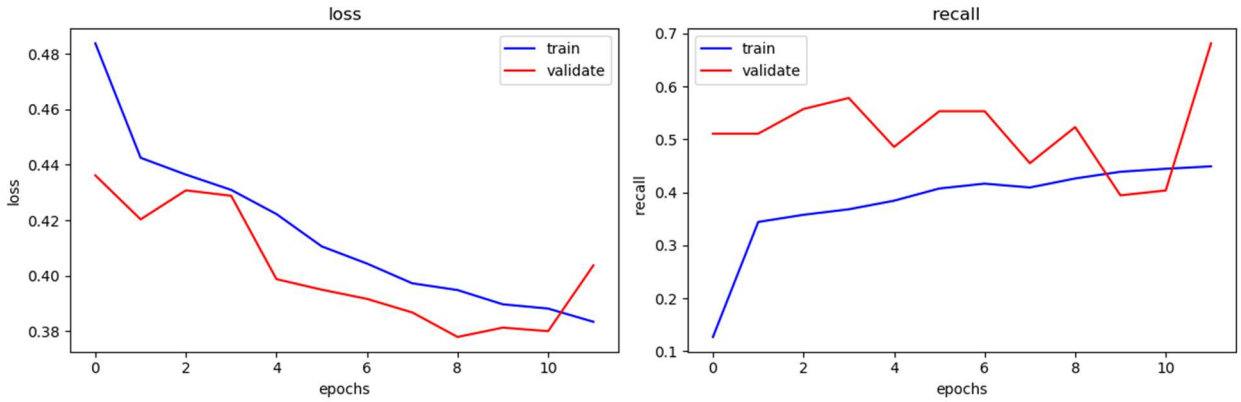


7. CNN- Improvements using PNG

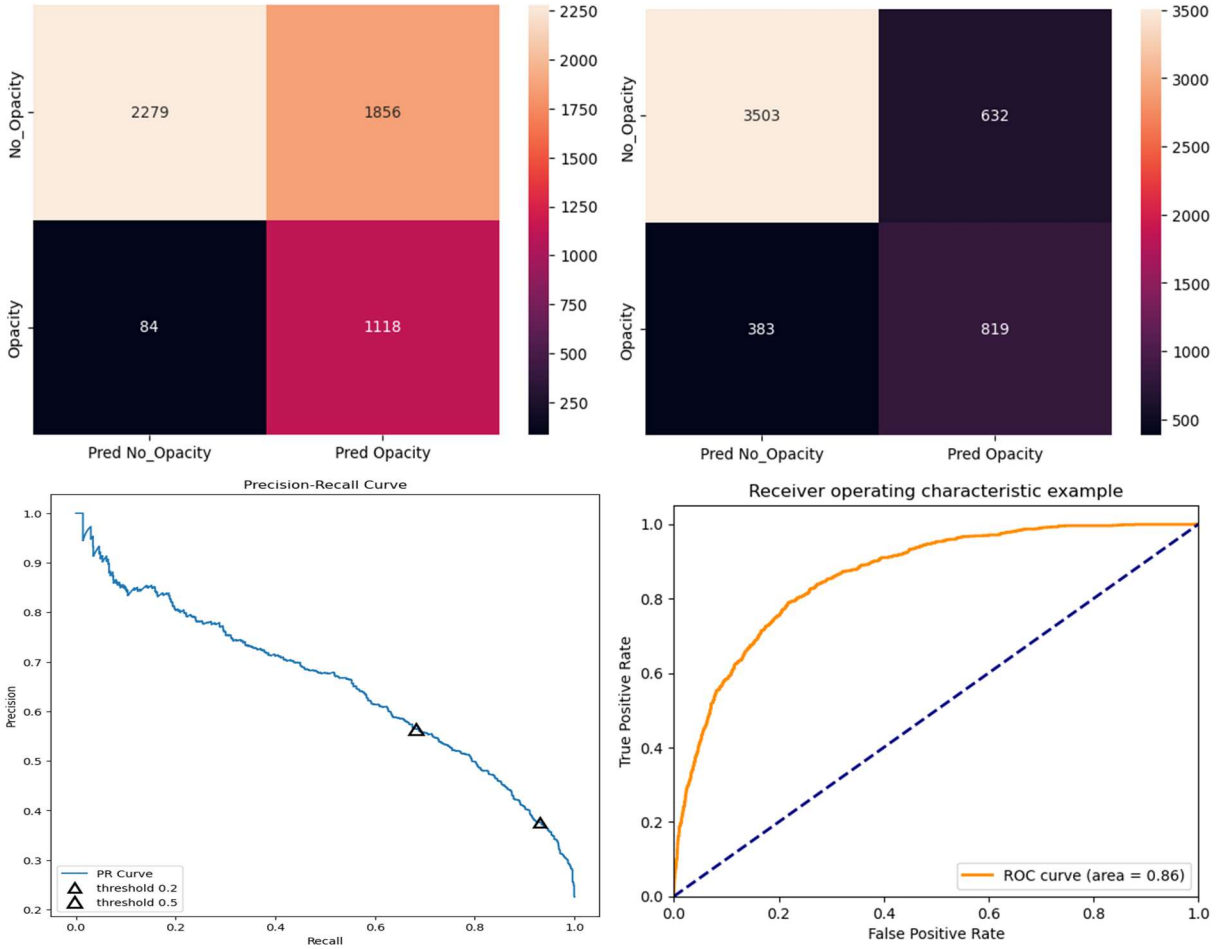
```
img_dims = image_size
inputs = Input(shape=(img_dims, img_dims, n_channels))

# First conv block
x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
```

```
x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Second conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Third conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)

# Fourth conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.3)(x)

# Fifth conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.4)(x)

# Sixth conv block
x = SeparableConv2D(filters=512, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = SeparableConv2D(filters=512, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.5)(x)

# FC layer
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(rate=0.7)(x)
x = Dense(units=256, activation='relu')(x)
x = Dropout(rate=0.5)(x)
x = Dense(units=128, activation='relu')(x)
x = Dropout(rate=0.3)(x)

# Output layer
output = Dense(units=num_classes, activation=actvn)(x)

# Creating model and compiling
model = tf.keras.Model(inputs=inputs, outputs=output)
model.compile(optimizer=Adam(learning_rate=0.0001), loss=lss, metrics=[Recall()])
```

```
model.summary()

Desc = 'CNN- Improvements using PNG'
Saved_Model = 'CNN- PNG 224-r.h5'
start_time = time()

# Callbacks
checkpoint = ModelCheckpoint(filepath=Saved_Model, save_best_only=True,
save_weights_only=True, verbose=1)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=2,
mode='min')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.0004, patience=5,
restore_best_weights = True)

history = model.fit(train_set,
                    validation_data = validation_set, epochs = EPOCHS_SET,
                    callbacks=[checkpoint, lr_reduce, early_stop])
end_time = time()
```




```
Threshold = 0.2:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.96      0.55      0.70      4135
     Opacity       0.38      0.93      0.54      1202

    accuracy                           0.64      5337
   macro avg       0.67      0.74      0.62      5337
weighted avg       0.83      0.64      0.66      5337

Threshold = 0.5:
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.90      0.85      0.87      4135
     Opacity       0.56      0.68      0.62      1202

    accuracy                           0.81      5337
   macro avg       0.73      0.76      0.75      5337
weighted avg       0.83      0.81      0.82      5337
```



8. MASK R-CNN based on Matterport Derived Implementation

MASK R-CNN is a popular instance segmentation model that builds upon the Faster R-CNN framework. The matterport implementation of MASK R-CNN is a widely used implementation that provides pre-trained models and code for training and inference.

The matterport implementation of MASK R-CNN is derived from the original implementation by researchers at Facebook AI Research. It includes improvements and optimizations to make it easier to use and perform better on various datasets.

With MASK R-CNN, you can not only detect objects in an image but also segment them by assigning pixel-level masks. This is useful for tasks such as object detection, semantic segmentation, and instance segmentation.

The matterport implementation provides a simple and straightforward interface for training. Mask R-CNN adopts the same two-stage procedure, with an identical first stage (which is RPN). In the second stage, in parallel to predicting the class and box offset, Mask R-CNN also outputs a binary mask for each RoI. This is in contrast to most recent systems, where classification depends on mask predictions

```
# Clone Mask_RCNN- Uncomment the below code appropriately to install the package and
weights

#!git clone https://www.github.com/leekunhee/Mask_RCNN.git

### Download COCO pre-trained weights
#import urllib.request

#url =
"https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5"
#filename = "mask_rcnn_coco.h5"
#urllib.request.urlretrieve(url, filename)


# Import Mask RCNN
sys.path.append(os.path.join(ROOT_DIR, MASK_RCNN_DIR))  # To find local version of the
library

from mrcnn.config import Config
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize
from mrcnn.model import log
```

preparing Input for Training and testing

```
def get_dicom_fps(dicom_dir):
    # Create a glob pattern for .dcm files in the directory
    dicom_fps = glob.glob(dicom_dir+'/'+'*.dcm')
    # Return a list of file paths, removing any duplicates by converting to a set
first
    return list(set(dicom_fps))

def parse_dataset(dicom_dir, anns):
    # Get a list of file paths for DICOM files in the directory
    image_fps = get_dicom_fps(dicom_dir)

    # Create a dictionary to hold the annotations for each image file
    # The key is the file path, and the value is an initially empty list
    image_annotations = {fp: [] for fp in image_fps}

    # For each annotation in the input DataFrame
    for index, row in anns.iterrows():
        # Create the file path for the associated image file
        fp = os.path.join(dicom_dir, row['patientId']+'.dcm')
        # Append the annotation to the list for this file path in the dictionary
        image_annotations[fp].append(row)

    # Return the list of file paths and the dictionary of annotations
    return image_fps, image_annotations

# get image file path and corresponding annotations
image_fps, image_annotations = parse_dataset(train_dicom_dir, anns=anns)

def dataset_model(image_names, target_values):
    """
    Splits the given image names and target values into train, validation, and test
sets.
    """

    # Split data into training and validation datasets
    # We use stratified sampling to ensure that the proportion of each class in both
datasets is the same
```

```
    # as in the original dataset
    image_fps_train, image_fps_val, targets_train, targets_val = train_test_split(
        image_names,
        target_values,
        stratify=target_values,
        shuffle=True,
        test_size=0.20,
        random_state=42
    )

    # Further split the training dataset into training and test datasets
    image_fps_train, image_fps_test, _, _ = train_test_split(
        image_fps_train,
        targets_train,
        stratify=targets_train,
        shuffle=True,
        test_size=0.20,
        random_state=42
    )

    # Append the directory path to the filenames
    image_fps_train = [os.path.join(train_dicom_dir, x + '.dcm') for x in
image_fps_train]
    image_fps_val = [os.path.join(train_dicom_dir, x + '.dcm') for x in image_fps_val]
    image_fps_test = [os.path.join(train_dicom_dir, x + '.dcm') for x in
image_fps_test]

    # Print the number of images in each set
    print(len(image_fps_train), len(image_fps_val), len(image_fps_test))

    return image_fps_train, image_fps_val, image_fps_test

# Let's assume image_fps_train, image_fps_val, image_fps_test are your lists

sample_train = image_fps_train
sample_val = image_fps_val
sample_test = image_fps_test
```

Define Hyper Parameters

```
# Define a dictionary with hyperparameters combinations for model tuning
hyper_paramters_comb = {
    'backbone': ['resnet50'],  # Backbone model to use
    'learning_rate': [0.004],  # Learning rate
    'batch_size': [batch_size],  # Batch size for training
    'epochs': [EPOCHS_SET],  # Number of epochs for training
    'det_min_conf': [0.9],  # Minimum confidence threshold for detection
    'det_nms_th': [0.6],  # Non-maximum suppression threshold for detection
    'rpn_nms_th': [0.5],  # Non-maximum suppression threshold for Region Proposal
Network
    'steps_per_epoch': [num_train//batch_size],  # Number of steps per epoch
    'layers': ['heads']  # Which layers to train
}

# Convert the dictionary to a pandas DataFrame
hpc = pd.DataFrame(hyper_paramters_comb)
```

Prepare test and validation dataset

```
from PIL import Image
class DetectorDataset(utils.Dataset):
    """Dataset class for training pneumonia detection on the RSNA pneumonia dataset.
```

```
    """

    def __init__(self, image_fps, image_annotations, orig_height, orig_width):
        super().__init__(self)

        # Add classes
        self.add_class('pneumonia', 1, 'Lung Opacity')

        # add images
        for i, fp in enumerate(image_fps):
            annotations = image_annotations[fp]
            self.add_image('pneumonia', image_id=i, path=fp,
                           annotations=annotations, orig_height=orig_height,
orig_width=orig_width)

    def image_reference(self, image_id):
        info = self.image_info[image_id]
        return info['path']

    def load_image(self, image_id):
        info = self.image_info[image_id]
        fp = info['path']
        ds = pydicom.read_file(fp)
        image = Image.fromarray(ds.pixel_array).convert('RGB')#ds.pixel_array

        # Resize the image
        img_dims=(info['orig_height'], info['orig_height'])
        image = image.resize(img_dims)
        image = image - np.min(image)
        return image

    def load_mask(self, image_id):
        info = self.image_info[image_id]
        annotations = info['annotations']
        count = len(annotations)
        if count == 0:
            mask = np.zeros((info['orig_height'], info['orig_width'], 1),
dtype=np.uint8)
            class_ids = np.zeros((1,), dtype=np.int32)
        else:
            mask = np.zeros((info['orig_height'], info['orig_width'], count),
dtype=np.uint8)
            class_ids = np.zeros((count,), dtype=np.int32)
            for i, a in enumerate(annotations):
                if a['Target'] == 1:
                    x = int(a['x']*(info['orig_height']/ORIG_SIZE))
                    y = int(a['y']*(info['orig_height']/ORIG_SIZE))
                    w = int(a['width']*(info['orig_height']/ORIG_SIZE))
                    h = int(a['height']*(info['orig_height']/ORIG_SIZE))
                    mask_instance = mask[:, :, i].copy()
                    cv2.rectangle(mask_instance, (x, y), (x+w, y+h), 255, -1)
                    mask[:, :, i] = mask_instance
                    class_ids[i] = 1
        return mask.astype(bool), class_ids.astype(np.int32)

# prepare the training dataset
dataset_train = DetectorDataset(sample_train, image_annotations, 256, 256)
dataset_train.prepare()

# prepare the validation dataset
dataset_val = DetectorDataset(sample_val, image_annotations, 256, 256)
dataset_val.prepare()
```

Image Augmentation

Image augmentation in deep learning refers to the technique of artificially increasing the size of a training dataset by applying various transformations to the existing images. These transformations can include rotation, scaling, flipping, cropping, and adding noise to the images. The main goal of image augmentation is to increase the diversity and variability of the training data, which helps the deep learning model to generalize better and improve its performance.

By applying image augmentation techniques, we can create more variations of the input images, allowing the model to learn different patterns and features. This helps in reducing overfitting and improves the model's ability to recognize and classify new, unseen images accurately.

```
# Define image augmentation sequence
augmentation = iaa.Sequential([

    # One of the following geometric transformations:
    iaa.OneOf([
        # Affine transformation: includes scaling, translating, rotating, and shearing
        iaa.Affine(
            scale={"x": (0.98, 1.02), "y": (0.98, 1.04)},  # Scale images
            translate_percent={"x": (-0.02, 0.02), "y": (-0.04, 0.04)},  # Translate
images
            rotate=(-10, 10),  # Rotate images
            shear=(-1, 1),  # Shear images
        ),
        # Piecewise affine transformation: A flexible local deformation of images
        #iaa.PiecewiseAffine(scale=(0.001, 0.025)),
    ]),
])
# Test the image augmentation on a single image.
# Draw a grid of several versions of the image with different augmentations
imggrid = augmentation.draw_grid(image[:, :, 0], cols=5, rows=2)

# Display the grid
plt.figure(figsize=(15, 6))
_ = plt.imshow(imggrid[:, :, 0], cmap='gray')
```

Detector Configuration

```
# Configuration class for pneumonia detection on the RSNA pneumonia dataset
class DetectorConfig(Config):
    """Configuration for training pneumonia detection on the RSNA pneumonia dataset.
    Overrides values in the base Config class.
    """

    # Give the configuration a recognizable name
    NAME = 'pneumonia'

    # Set the GPU count and images per GPU for training
    GPU_COUNT = 1
    IMAGES_PER_GPU = batch_size

    # Set the backbone model and batch size
    BACKBONE = hpc.iloc[0]['backbone']
    BATCH_SIZE = hpc.iloc[0]['batch_size']

    # Set the number of classes: background + 1 pneumonia class
    NUM_CLASSES = 2

    # Set whether to use mini-mask
```

```
    USE_MINI_MASK = False

    # Set the minimum and maximum image dimensions
    IMAGE_MIN_DIM = 256
    IMAGE_MAX_DIM = 256

    LOSS_WEIGHTS = {'rpn_class_loss': 1.0,
                    'rpn_bbox_loss': 1.0,
                    'mrcnn_class_loss': 1.25,
                    'mrcnn_bbox_loss': 1.0,
                    'mrcnn_mask_loss': 1.0}

    # Set the scales for the region proposal network (RPN) anchors
#    RPN_ANCHOR_SCALES = (32, 64, 128) #, 256
#    RPN_TRAIN_ANCHORS_PER_IMAGE = 32

    # Set the number of ROIs per image for training
    TRAIN_ROIS_PER_IMAGE = 8
    # Set the maximum ground truth and detection instances
    MAX_GT_INSTANCES = 4
    DETECTION_MAX_INSTANCES = 3

    # Minority class ratio
    ROI_POSITIVE_RATIO = 0.225

    # Set the minimum detection confidence
    DETECTION_MIN_CONFIDENCE = hpc.iloc[0]['det_min_conf']

    # Set the detection and RPN NMS threshold
    DETECTION_NMS_THRESHOLD = hpc.iloc[0]['det_nms_th']
    RPN_NMS_THRESHOLD = hpc.iloc[0]['rpn_nms_th']

    # Set the steps per epoch
    STEPS_PER_EPOCH = hpc.iloc[0]['steps_per_epoch']

# Create an instance of the configuration class
config = DetectorConfig()

# Display the configuration
config.display()
```

Load pre-trained Weights

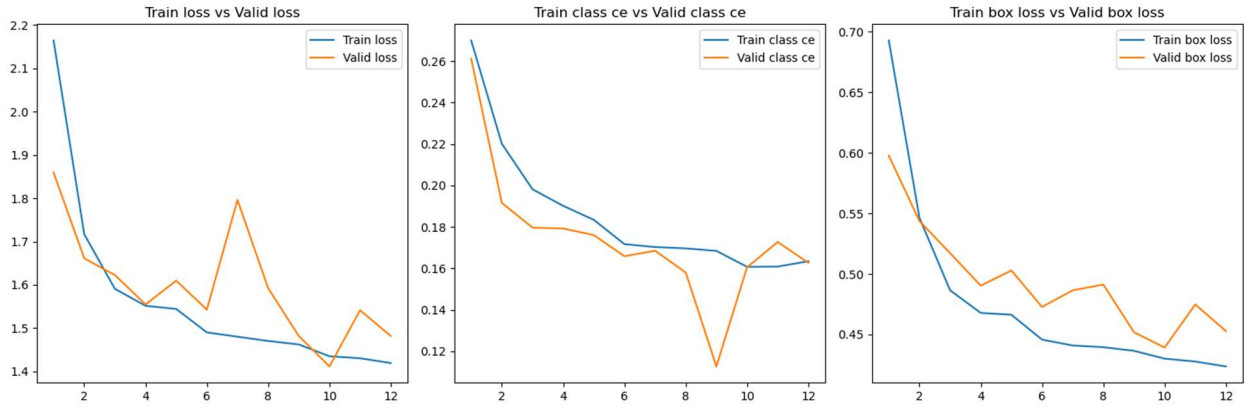
```
# Instantiate the MaskRCNN model in training mode using the given configurations and
model directory
model = modellib.MaskRCNN(mode='training', config=config, model_dir=ROOT_DIR)

# Load the weights from the COCO dataset, excluding the last layers
# This is because the last layers require a matching number of classes
model.load_weights(COCO_WEIGHTS_PATH, by_name=True, exclude=[
    "mrcnn_class_logits", "mrcnn_bbox_fc", "mrcnn_bbox", "mrcnn_mask"])
```

Set check points for saving model after each epoch

```
# Define the path where the checkpoints will be saved
checkpoint_path = os.path.join(ROOT_DIR,
"mask_rcnn_{}_*epoch*.h5".format(config.NAME.lower()))

# Format the checkpoint file name to include the epoch number
checkpoint_path = checkpoint_path.replace("*epoch*", "{epoch:04d}")
```

```
# Set up a callback to save the model's weights after each epoch
callbacks = [keras.callbacks.ModelCheckpoint(checkpoint_path, verbose=1,
                                             save_weights_only=True,
save_freq='epoch')]

Train the Mask R CNN Model
%%time
# Train the model on the dataset
model.train(dataset_train, dataset_val,
            learning_rate=hpc.iloc[0]['learning_rate'],
            epochs=hpc.iloc[0]['epochs'],
            custom_callbacks=callbacks,
            layers=hpc.iloc[0]['layers'],
            augmentation=augmentation)

# Save the history of the training for future analysis
history = model.keras_model.history.history
```

Results



Load the model in inference mode

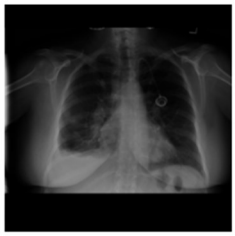
```
# Create an inference configuration class that inherits from the DetectorConfig
class InferenceConfig(DetectorConfig):
    # Set the GPU count to 1
    GPU_COUNT = 1
    # Set the number of images per GPU to 1
    IMAGES_PER_GPU = 1

# Instantiate the InferenceConfig
inference_config = InferenceConfig()

# Import necessary modules
from keras.backend import manual_variable_initialization
import tensorflow.compat.v1 as tf

# Set manual variable initialization to True
manual_variable_initialization(True)

# Recreate the model in inference mode
model = modellib.MaskRCNN(mode="inference",
                          model_dir='.',
                          config=inference_config)

# Ensure that the model path is not empty
```

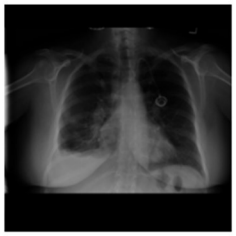
```
assert model_path != "", "Provide path to trained weights"
print("Loading weights from ", model_path)

# Load the trained weights into the model
tf.keras.Model.load_weights(model.keras_model, model_path, by_name=True)

# Prepare the test dataset
dataset_test = DetectorDataset(sample_test, image_annotations, 256, 256)
dataset_test.prepare()

# Subset the dataset to the first 1000 images
images = dataset_test.image_ids

# Initialize empty lists for true and predicted bounding boxes
ytrue = []
ypred = []

# Iterate over the images
for i in range(len(images)):
    # Load the image along with its ground truth bounding boxes and masks
    original_image, _, _, gt_bbox, gt_mask = modellib.load_image_gt(dataset_test,
inference_config, images[i])

    # Append the ground truth bounding boxes to the true list
    ytrue.append(gt_bbox)

    # Detect objects in the image and get the predicted bounding boxes
    results = model.detect([original_image])

    # Append the predicted bounding boxes to the prediction list
    ypred.append(results[0]['rois'])
```

Visualize the results

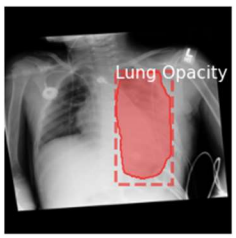
Observations

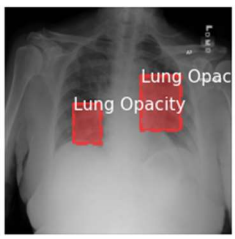
Images Ground truth (Left side Images above) Predicted (Right side images) indicate that in 5 out of 6 cases the model correctly predicted the ground truth. Errors are expected as large number of X-rays are for **not normal** cases which have features similar to **opacity / pneumonia** cases. Tuning efforts has resulted in improvement of recall



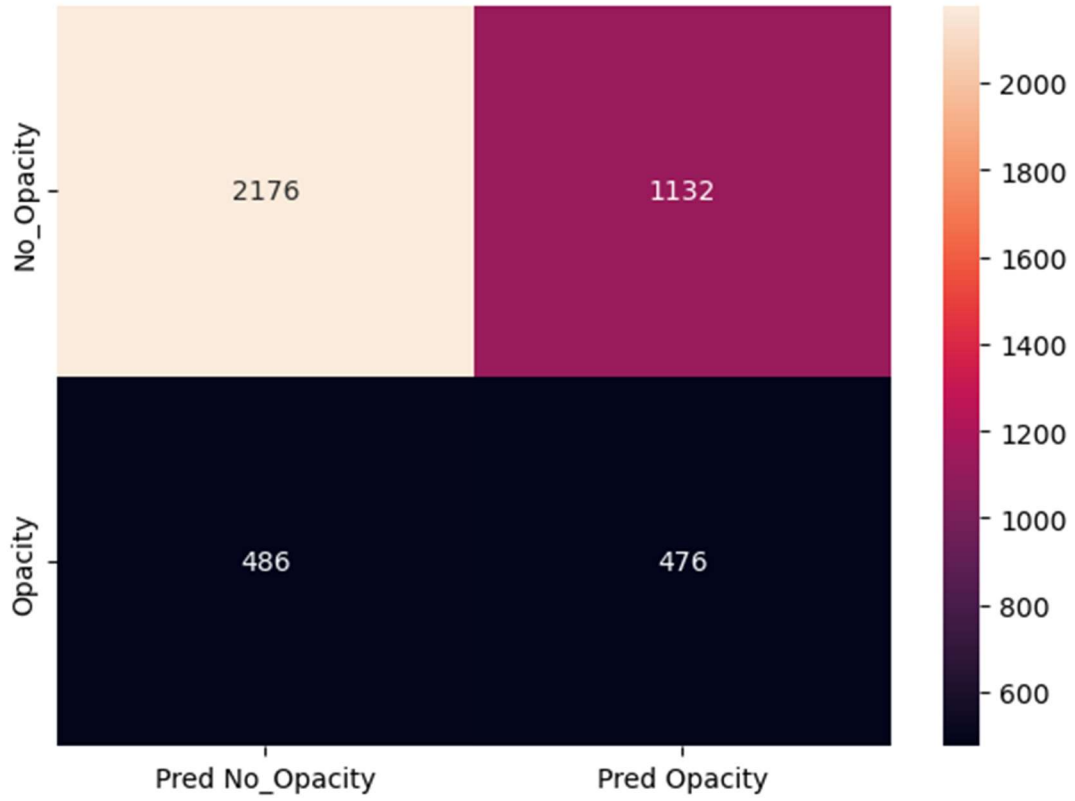



Lung Opacity


Lung Opacity


Lung Opac
Lung Opacity

```
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.82      0.66      0.73      3308
     Opacity       0.30      0.49      0.37       962

    accuracy                           0.62      4270
   macro avg       0.56      0.58      0.55      4270
weighted avg       0.70      0.62      0.65      4270
```



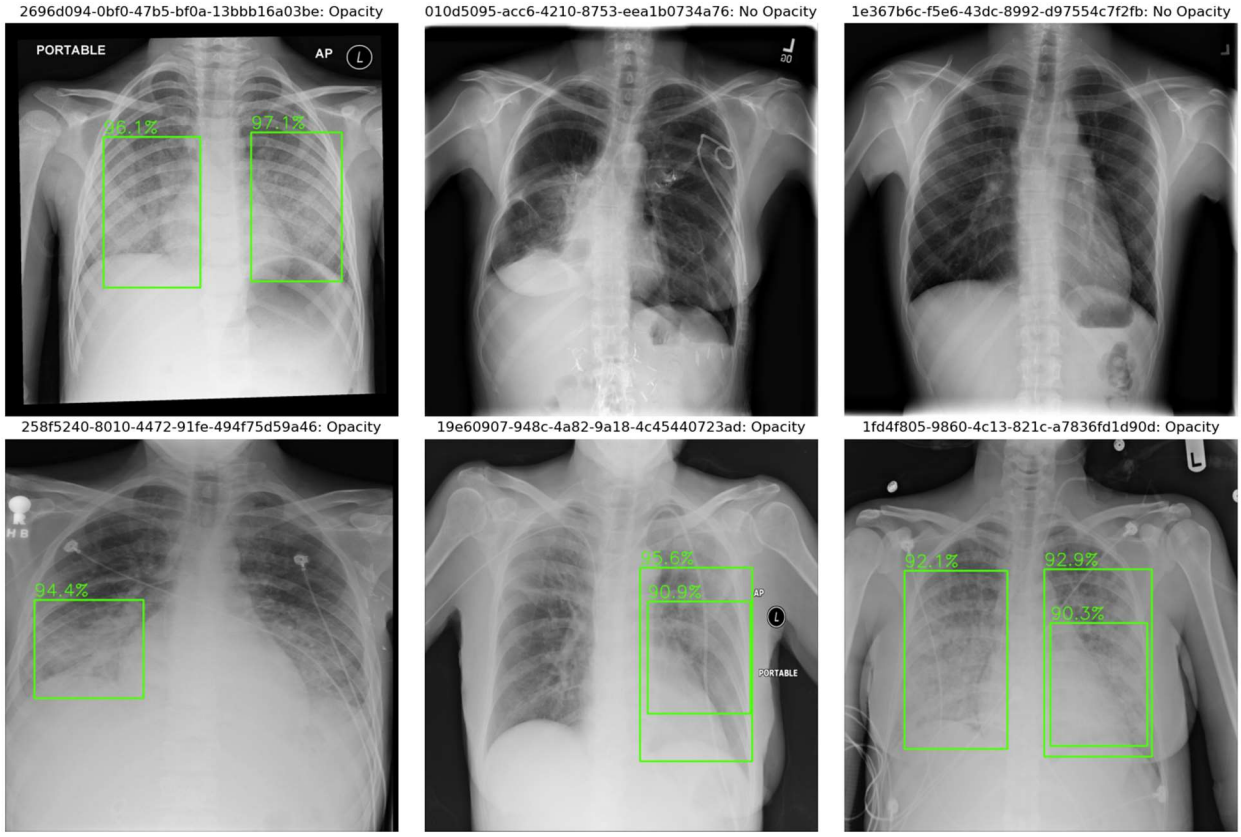
Final steps to create Submission File

```
# Set the submission file path
submission_fp = os.path.join(ROOT_DIR, 'submission.csv')

# Run predictions on the test images and create a submission file
predict(test_image_fps, filepath=submission_fp)
```

Visualize test Image predicttion

```
def visualize_results(image_id, ax):
    ds = pydicom.read_file(image_id)

    # original image
    # If grayscale. Convert to RGB for consistency.
    image = Image.fromarray(ds.pixel_array).convert('RGB')

    # Convert the PIL image to a NumPy array
```

```
    image = np.array(image)

    # assume square image
    resize_factor = ORIG_SIZE / config.IMAGE_SHAPE[0]
    # Resize the image
    resized_image, window, scale, padding, crop = utils.resize_image(
        image,
        min_dim=config.IMAGE_MIN_DIM,
        min_scale=config.IMAGE_MIN_SCALE,
        max_dim=config.IMAGE_MAX_DIM,
        mode=config.IMAGE_RESIZE_MODE)

    resized_image = resized_image - np.min(resized_image)
    patient_id = os.path.splitext(os.path.basename(image_id))[0]

    results = model.detect([resized_image])
    r = results[0]

    for i, bbox in enumerate(r['rois']):
        x1 = int(bbox[1] * resize_factor)
        y1 = int(bbox[0] * resize_factor)
        x2 = int(bbox[3] * resize_factor)
        y2 = int(bbox[2] * resize_factor)
        cv2.rectangle(image, (x1,y1), (x2,y2), (77, 255, 9), 3, 1)

        # Draw the confidence score above the bounding box
        score = r['scores'][i]
        confidence_text = f"{score*100:.1f}%"
        cv2.putText(image, confidence_text, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX,
1.5, (77, 255, 9), 2)

    # Display image with title
    title = f'{patient_id}: {"Opacity" if len(r["rois"]) > 0 else "No Opacity"}'
    ax.set_title(title)
    ax.imshow(image, cmap=plt.cm.gist_gray)
    ax.axis('off')  # Hide axis

def display_test_images(n_img, n_col):
    # calculate the number of rows needed
    n_rows = math.ceil(n_img / n_col)

    fig, axes = plt.subplots(n_rows, n_col, figsize=(n_col * 5, n_rows * 5))
    axes = axes.ravel()  # flatten axes for easier iteration

    for i in range(n_img):
        image_id = random.choice(test_image_fps)
        visualize_results(image_id, axes[i])

    # Remove unused subplots
    if n_img % n_col != 0:
        for i in range(n_img, n_rows * n_col):
            fig.delaxes(axes[i])

    plt.tight_layout()
    plt.show()

# Show 10 images in a grid
display_test_images(6, 3)
```



Pickle the model for Future Prediction

To pickle the model for future predictions, you can use the pickle module in Python. Pickling allows you to serialize the trained model and save it to a file, which can then be loaded and used for making predictions in the future.

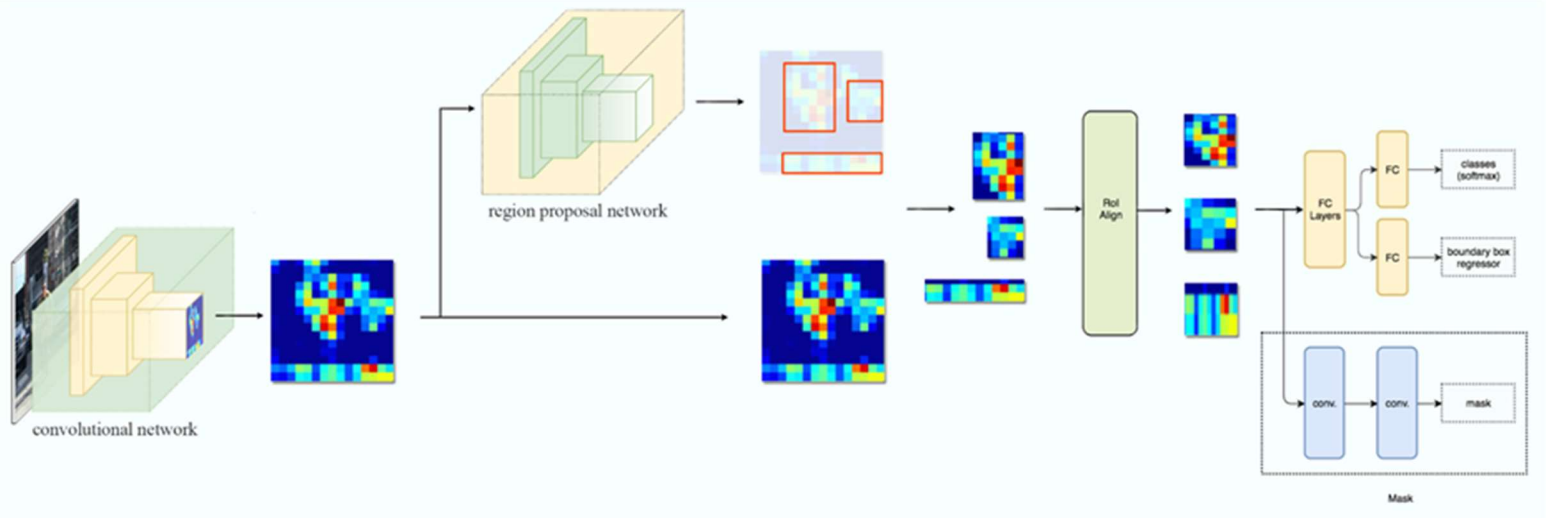
```
# Pickle the best Mask R-CNN model
import pickle

pickle_fn = 'Best MRCNN.pkl'

pickle.dump(model, open(pickle_fn, 'wb'))
```

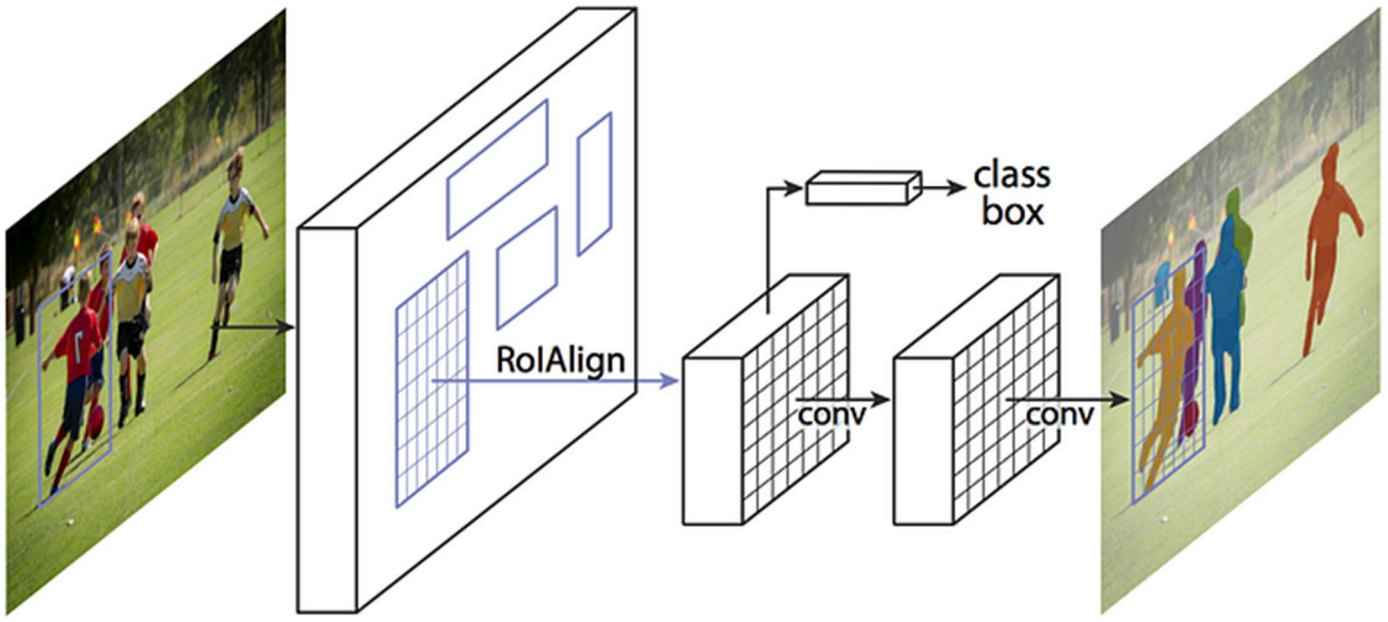
4 Model evaluation for Mask R-CNN

Description of the **Mask R-CNN** model:

Mask R-CNN (regional convolutional neural network) is a two stage framework: the first stage scans the image and generates *proposals*(areas likely to contain an object). And the second stage classifies the proposals and generates bounding boxes and masks.

It was introduced last year via the Mask R-CNN paper to extend its predecessor, Faster R-CNN, by the same authors. Faster R-CNN is a popular framework for object detection, and Mask R-CNN extends it with instance segmentation, among other things.

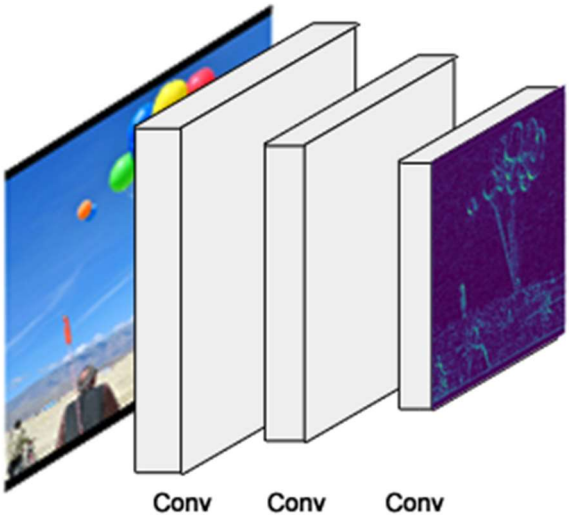




Mask R-CNN framework. Source: https://arxiv.org/abs/1703.06870

At a high level, Mask R-CNN consists of these modules:

1. Backbone

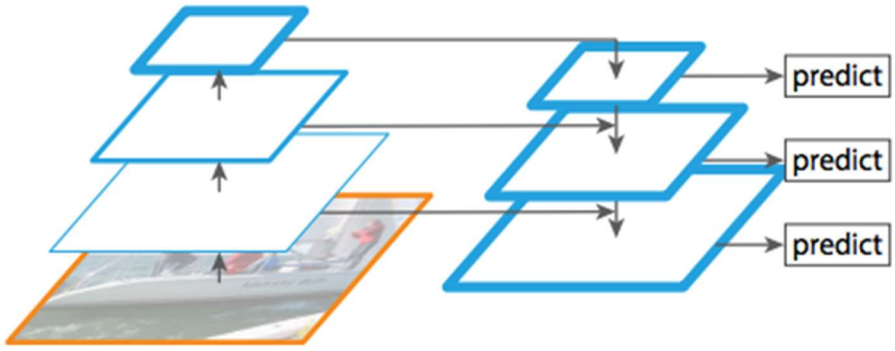


Simplified illustration of the backbone nework

This is a standard convolutional neural network (typically, ResNet50 or ResNet101) that serves as a feature extractor. The early layers detect low level features (edges and corners), and later layers successively detect higher level features (car, person, sky).

Passing through the backbone network, the image is converted from 1024x1024px x 3 (RGB) to a feature map of shape 32x32x2048. This feature map becomes the input for the following stages.

Feature Pyramid Network



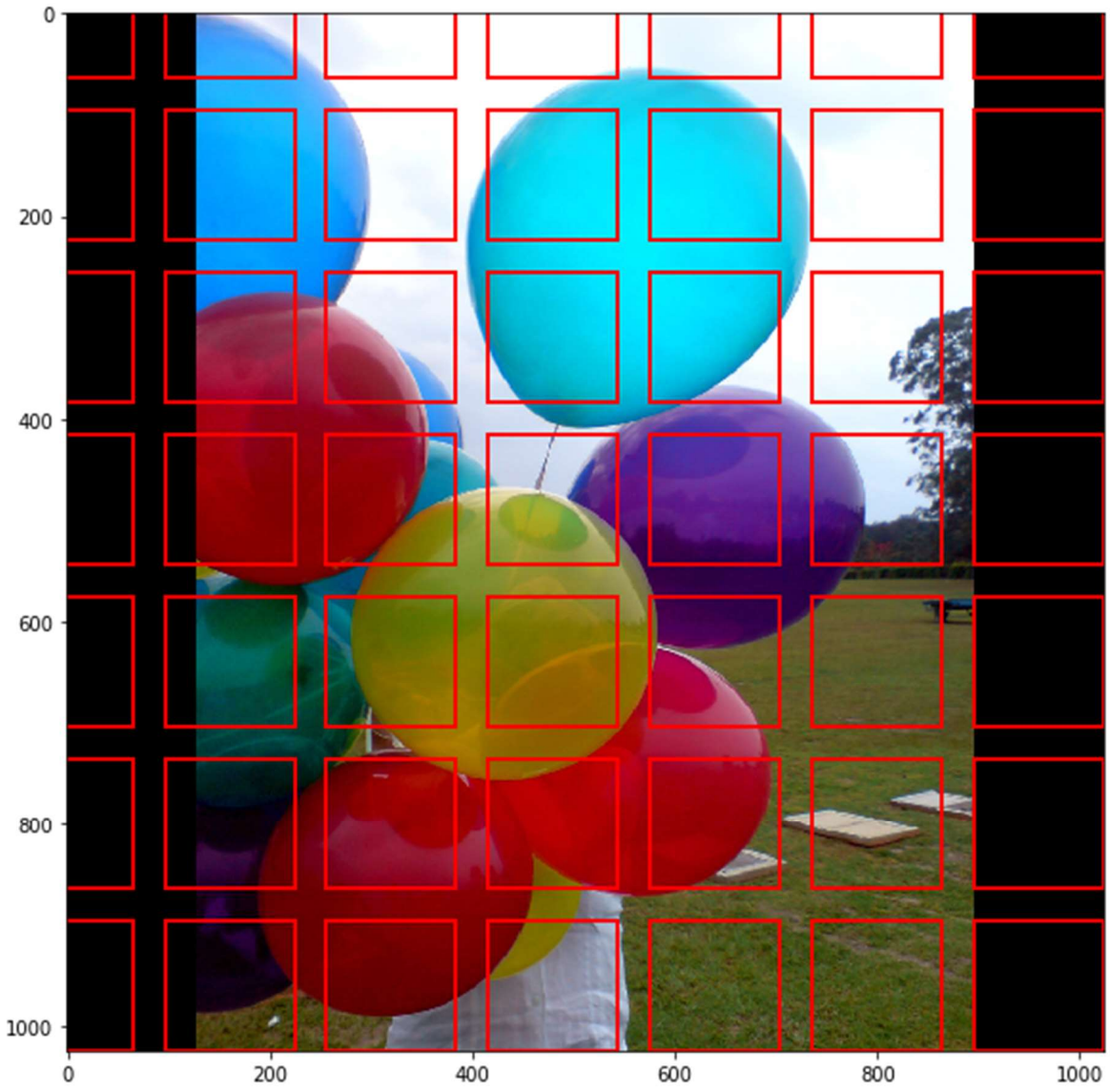
Source: Feature Pyramid Networks paper

While the backbone described above works great, it can be improved upon. The Feature Pyramid Network (FPN) was introduced by the same authors of Mask R-CNN as an extension that can better represent objects at multiple scales.

FPN improves the standard feature extraction pyramid by adding a second pyramid that takes the high level features from the first pyramid and passes them down to lower layers. By doing so, it allows features at every level to have access to both, lower and higher level features.

Our implementation of Mask RCNN uses a ResNet101 + FPN backbone.

2. Region Proposal Network (RPN)



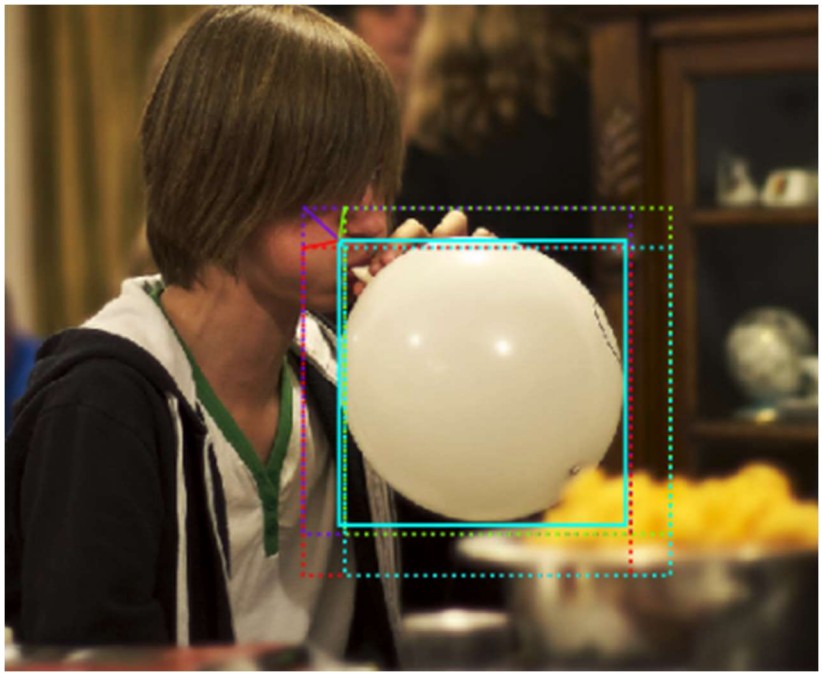
Simplified illustration showing 49 anchor boxes

The RPN is a lightweight neural network that scans the image in a sliding-window fashion and finds areas that contain objects.

The regions that the RPN scans over are called *anchors*. Which are boxes distributed over the image area, as show on the left. This is a simplified view, though. In practice, there are about 200K anchors of different sizes and aspect ratios, and they overlap to cover as much of the image as possible.

How fast can the RPN scan that many anchors? Pretty fast, actually. The sliding window is handled by the convolutional nature of the RPN, which allows it to scan all regions in parallel (on a GPU). Further, the RPN doesn't scan over the image directly (even though we draw the anchors on the image for illustration). Instead, the RPN scans over the backbone feature map. This allows the RPN to reuse the extracted features efficiently and avoid duplicate calculations. With these optimizations, the RPN runs in about 10 ms according to the Faster RCNN paper that introduced it. In Mask RCNN we typically use larger images and more anchors, so it might take a bit longer.

The RPN generates two outputs for each anchor:



3 anchor boxes (dotted) and the shift/scale applied to them to fit the object precisely (solid). Several anchors can map to the same object.

1. **Anchor Class:** One of two classes: foreground or background. The FG class implies that there is likely an object in that box.
2. **Bounding Box Refinement:** A foreground anchor (also called positive anchor) might not be centered perfectly over the object. So the RPN estimates a delta (% change in x, y, width, height) to refine the anchor box to fit the object better.

Using the RPN predictions, we pick the top anchors that are likely to contain objects and refine their location and size. If several anchors overlap too much, we keep the one with the highest foreground score and discard the rest (referred to as Non-max Suppression). After that we have the final *proposals* (regions of interest) that we pass to the next stage.

3. ROI Classifier & Bounding Box Regressor

This stage runs on the regions of interest (ROIs) proposed by the RPN. And just like the RPN, it generates two outputs for each ROI:

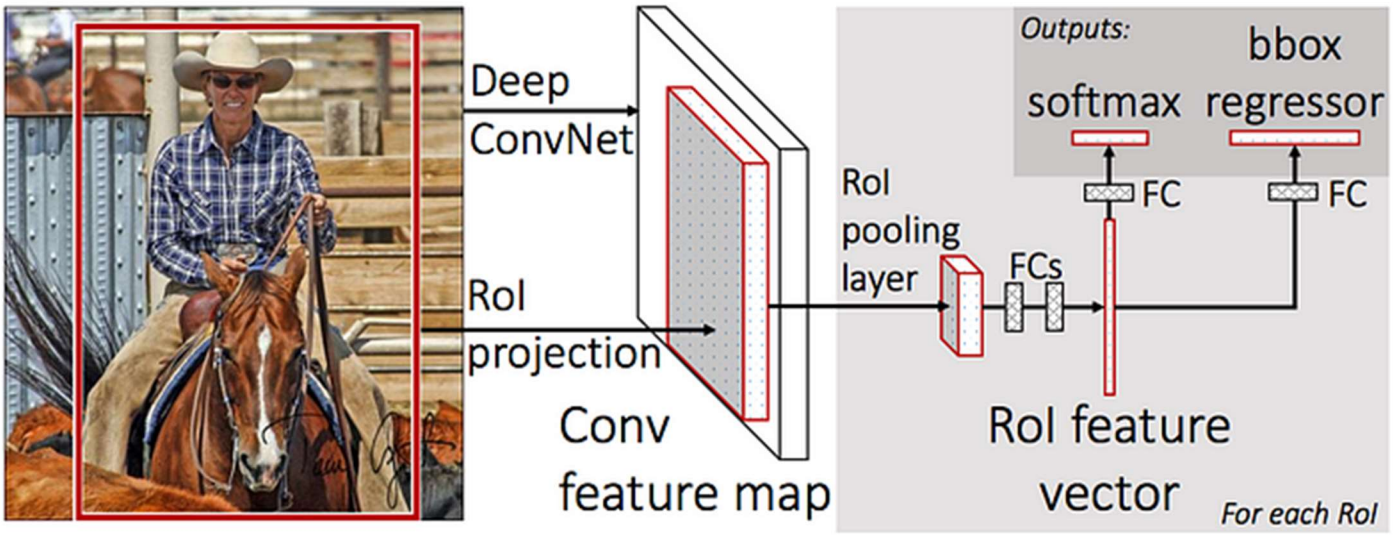
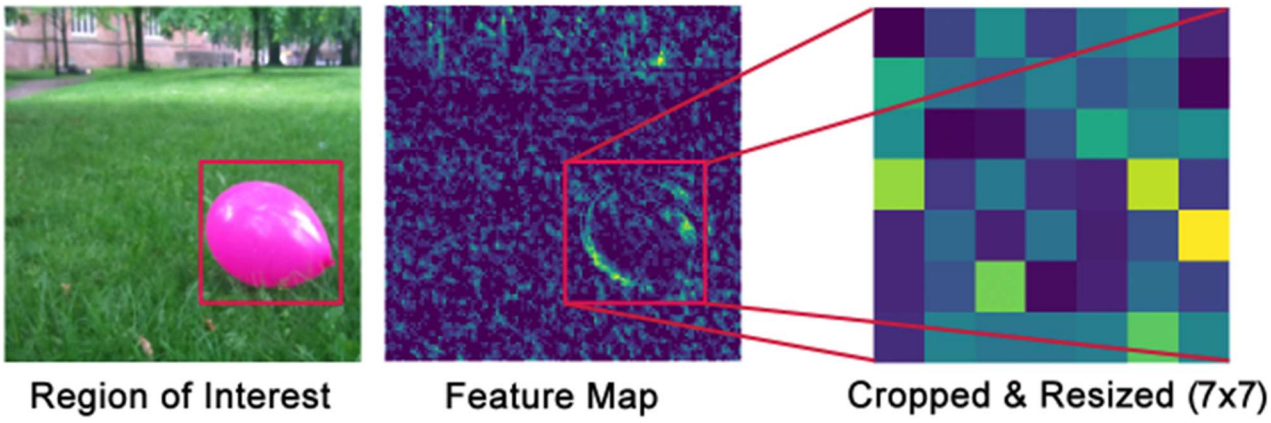


Illustration of stage 2. Source: Fast R-CNN (https://arxiv.org/abs/1504.08083)

1. **Class:** The class of the object in the ROI. Unlike the RPN, which has two classes (FG/BG), this network is deeper and has the capacity to classify regions to specific classes (person, car, chair, …etc.). It can also generate a *background* class, which causes the ROI to be discarded.
2. **Bounding Box Refinement:** Very similar to how it's done in the RPN, and its purpose is to further refine the location and size of the bounding box to encapsulate the object.

ROI Pooling

There is a bit of a problem to solve before we continue. Classifiers don't handle variable input size very well. They typically require a fixed input size. But, due to the bounding box refinement step in the RPN, the ROI boxes can have different sizes. That's where ROI Pooling comes into play.



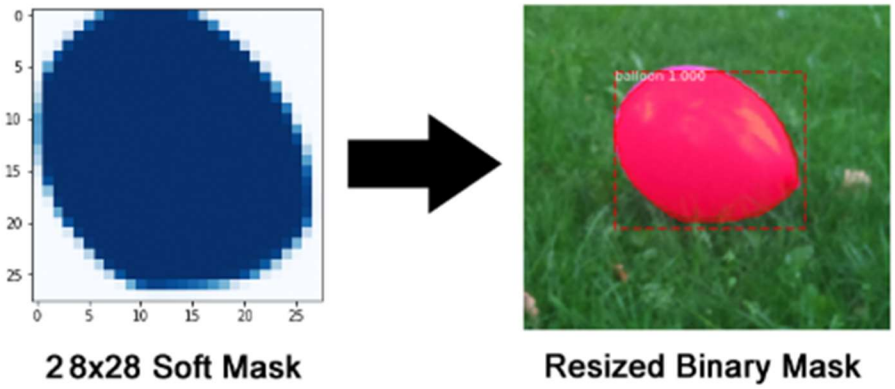
The feature map here is from a low-level layer, for illustration, to make it easier to understand.

ROI pooling refers to cropping a part of a feature map and resizing it to a fixed size. It's similar in principle to cropping part of an image and then resizing it (but there are differences in implementation details).

The authors of Mask R-CNN suggest a method they named ROIAlign, in which they sample the feature map at different points and apply a bilinear interpolation. In our implementation, we used TensorFlow's crop_and_resize function for simplicity and because it's close enough for most purposes.

4. Segmentation Masks

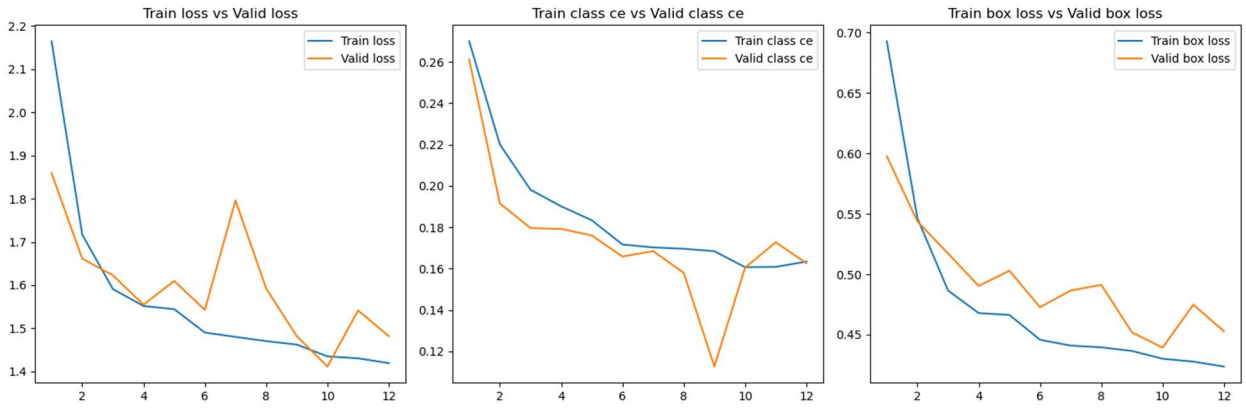
If you stop at the end of the last section then you have a Faster R-CNN framework for object detection. The mask network is the addition that the Mask R-CNN paper introduced.



The mask branch is a convolutional network that takes the positive regions selected by the ROI classifier and generates masks for them. The generated masks are low resolution: 28x28 pixels. But they are *soft* masks, represented by float numbers, so they hold more details than binary masks. The small mask size helps keep the mask branch light. During training, we scale down the ground-truth masks to 28x28 to compute the loss, and during inferencing we scale up the predicted masks to the size of the ROI bounding box and that gives us the final masks, one per object.

(Ref: https://engineering.matterport.com/)

Mask R-CNN Training & Validation losses indicating Properly trained model



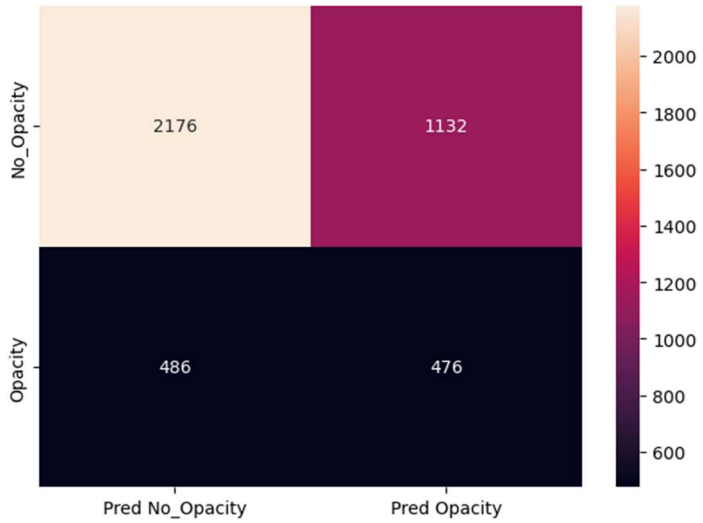
```
# Define the range of epochs combining history over all runs
epochs = range(1, len(next(iter(history.values())))+1)

# Create a pandas DataFrame from the history dictionary
Results_MaskRCNN = pd.DataFrame(history, index=epochs)
Results_MaskRCNN
```

	loss	rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss
1	2.163973	0.089802	0.630541	0.269921	0.692802	0.480907	1.859143	0.092488	0.507191	0.261142	0.597751	0.400570
2	1.716790	0.062483	0.449295	0.220121	0.546375	0.438516	1.661175	0.059979	0.457907	0.191595	0.543698	0.407996
3	1.590891	0.056026	0.422739	0.198175	0.486516	0.427436	1.622680	0.064337	0.481100	0.179657	0.517197	0.380389
4	1.551704	0.054076	0.418414	0.190186	0.467800	0.421229	1.554842	0.051831	0.448617	0.179242	0.490333	0.384820
5	1.544423	0.054894	0.420934	0.183441	0.466310	0.418845	1.609771	0.057631	0.488763	0.176077	0.502846	0.384454
6	1.490315	0.051377	0.409387	0.171702	0.445750	0.412099	1.542837	0.051433	0.447244	0.165891	0.472731	0.405537
7	1.480336	0.050328	0.407394	0.170309	0.440914	0.411390	1.796122	0.067321	0.689499	0.168526	0.486463	0.384313
8	1.470508	0.049875	0.403510	0.169666	0.439495	0.407962	1.592363	0.057538	0.462251	0.158030	0.491213	0.423331
9	1.462488	0.049638	0.401283	0.168468	0.436484	0.406615	1.481858	0.062966	0.469181	0.112723	0.451856	0.385133
10	1.435226	0.048205	0.393768	0.160772	0.430036	0.402445	1.411709	0.048679	0.399495	0.160526	0.439216	0.363792
11	1.430653	0.047413	0.391872	0.160892	0.427654	0.402821	1.541501	0.048801	0.444857	0.172765	0.474816	0.400263
12	1.419619	0.046611	0.386433	0.163455	0.423584	0.399537	1.482012	0.051488	0.426041	0.162719	0.452825	0.388938

Mask R-CNN Performance parameters on Validation Data

```
Classification Report
              precision    recall  f1-score   support

  No_Opacity       0.82      0.66      0.73      3308
     Opacity       0.30      0.49      0.37       962

    accuracy                           0.62      4270
   macro avg       0.56      0.58      0.55      4270
weighted avg       0.70      0.62      0.65      4270
```

Confusion Matrix



Comparison of Performance parameters

```
results_df
```

	Description	Threshold	Accuracy	Precision	Recall	F1 Score	Model	Execution Time
0	CNN for classification - with recall	0.2	0.766429	0.480086	0.722853	0.576973	CNN- DICOM 224r.h5	560.580348
1	CNN for classification - with recall	0.5	0.811786	0.680000	0.275527	0.392157	CNN- DICOM 224r.h5	560.580348
2	CNN for classification - with recall & accuracy	0.2	0.530357	0.305246	0.886548	0.454130	CNN- DICOM 224-ra.h5	495.309162
3	CNN for classification - with recall & accuracy	0.5	0.798929	0.629808	0.212318	0.317576	CNN- DICOM 224-ra.h5	495.309162
4	VGG16 with recall & accuracy	0.2	0.756786	0.470205	0.818476	0.597280	VGG16- DICOM 224 ra.h5	524.290136
5	VGG16 with recall & accuracy	0.5	0.831071	0.673913	0.452188	0.541222	VGG16- DICOM 224 ra.h5	524.290136
6	VGG16 with recall	0.2	0.815357	0.564767	0.706645	0.627790	VGG16- DICOM 224 r.h5	521.236001
7	VGG16 with recall	0.5	0.842500	0.727979	0.455429	0.560319	VGG16- DICOM 224 r.h5	521.236001
8	Object Detection- Mask R-CNN	0.2	0.811429	0.561043	0.662885	0.607727	Mask R-CNN- DICOM 224.h5	541.458546
9	Object Detection- Mask R-CNN	0.5	0.818929	0.586751	0.602917	0.594724	Mask R-CNN- DICOM 224.h5	541.458546
10	VGG16- Improvements using PNG	0.2	0.750234	0.469407	0.836106	0.601256	VGG16- PNG 224-ra.h5	-35.298264
11	VGG16- Improvements using PNG	0.5	0.826682	0.669523	0.455075	0.541852	VGG16- PNG 224-ra.h5	-35.298264
12	CNN- Improvements using PNG	0.2	0.636500	0.375925	0.930116	0.535441	CNN- PNG 224-r.h5	1495.137465
13	CNN- Improvements using PNG	0.5	0.809818	0.564438	0.681364	0.617414	CNN- PNG 224-r.h5	1495.137465
14	Object Detection with bounding boxes- Mask R-CNN	0.5	0.621077	0.296020	0.494802	0.370428	mask_rcnn_pneumonia_XXXX.h5	17500.297880

The table presents the results from several machine learning models. These models have been trained and tested on different tasks (classification or object detection), using different architectures (CNN, VGG16, Mask R-CNN), and different image types (DICOM, PNG).

Performance metrics include accuracy, precision, recall, F1 score, and execution time. The threshold column likely refers to a decision threshold for classifying an instance to a particular class. Different thresholds yield different trade-offs between precision (how many selected items are relevant?) and recall (how many relevant items are selected?).

In general, increasing the threshold appears to result in a higher precision but lower recall. For example, consider the CNN for classification - with recall model. The precision increases from 0.48 to 0.68 as the threshold increases from 0.2 to 0.5, but the recall decreases from 0.72 to 0.28.

Interestingly, the performance of the same architecture can vary significantly depending on the input image type. Compare VGG16- Improvements using PNG and VGG16 with recall & accuracy. The accuracy is similar for both models, but the model trained on PNG images has a much lower execution time, which could be a critical factor in a real-time application.

Finally, the Mask R-CNN models show a significant increase in execution time, likely due to the more complex task of object detection (compared to image classification). The last row is especially interesting because even though the precision is low (0.219659), the recall is perfect (1.000000), implying the model identifies all relevant instances but also includes many irrelevant ones.

Hyper-parameters:

The following are few Hyper-parameters specific to Mask R-CNN. Important ones are marked bold.

- Back Bone – Dfault ResNet50 used
- Train_ROIs_Per_Image – default was over-ridden and 8 was used for efficiency purpose
- Max_GT_Instances – 4 selected as max 4 bounding boxes were noted in training images
- **Detection_Min_Confidence – This is the most important parameter which determines recall, accuracy.**
- Image_Min_Dim and Image_Max_Dim – 256x256 was used to speed up the training
- Loss Weights : rpn_class_loss – kept default value
- Loss Weights : rpn_bbox_loss– kept default value
- Loss Weights : mrcnn_class_loss – This parameter was increased to improve the performance parameters such as accuracy, precision and recall.
- Loss Weights : mrcnn_bbox_loss– kept default value
- Loss Weights : mrcnn_mask_loss– kept default value
- **Learning rate – Learning rate was increase to 0.04 for faster conversion, High value is justified as data augmentation was used.**
- Batch size for training – 16 was kept uniform for all models
- **Non-maximum suppression threshold for detection – This parameter is also important for achieving better performance parameters**
- **Non-maximum suppression threshold for Region Proposal Network – This parameter is also important for achieving better performance parameters**

5. Comparison to benchmark set in interim report base models

- **Discussion on the results:**
  Minority class, i.e., Opacity case performance parameters for **Mask R-CNN with bounding boxes**:
- **Precision = 0.30, Recall = 0.49, F1 Score = 0.37, Mean IOU = 0.57.**
- Minority class, i.e., Opacity case performance parameters for **Mask R-CNN with mask**:
- **Precision = 0.56 Recall = 0.66 F1 Score = 0.61**
- Which at the out set would look poor but when compared to the **base model** results from Milestone 1 the parameters were Opacity
- **Precision = 0.65 Recall = 0.34 F1 Score = 0.45**
- It is clear that we have achieved definite improvement in the **recall of minority class**. Further hyper parameter tuning will yield improve the recall of minority class. The hyper parameters which important in mask R – CNN are as follows

From the results above it is clear that we have achieved a marked improvement in performance parameters in all models in this submission when compared to the base models in interim report. **The success is evident when we compare the recall of minority class?**

6. Visualizations

Visualisations have been added at appropriate locations in the report

7. Implications

How does your solution affect the problem in the domain or business? What recommendations would you make, and with what level of confidence?

In medical diagnostics, especially in areas like pneumonia detection where a false negative can have serious health implications, achieving a high recall is often a primary goal. A high recall rate in the 'Opacity' class means that the model can successfully identify a high proportion of actual pneumonia cases.

Case 1: In this scenario, the recall for detecting 'Opacity', which indicates possible pneumonia, is relatively low at 0.30. This means that 70% of actual pneumonia cases were not identified by the model (false negatives), which could have serious health implications for those patients. The precision for detecting 'Opacity' is also relatively low (0.62), indicating that when the model predicts an 'Opacity' case, it is correct 62% of the time.

Case 2: In the second scenario, the model performance has improved. The recall for 'Opacity' is 0.50, which means the model is now correctly identifying half of the pneumonia cases. However, this also means that half of the pneumonia cases are still missed, leading to a significant number of false negatives. The precision for 'Opacity' is the same as in Case 1 (0.62).

In both cases, the models have better performance for the 'Normal'/'No_Opacity' class. This could be due to class imbalance in the dataset (if there are significantly more 'Normal' images than 'Opacity' images), or it could be that the models have a harder time identifying the features that indicate 'Opacity' in the images.

In conclusion, while the models demonstrate reasonable performance in classifying 'Normal'/'No_Opacity' images, they struggle with accurately identifying 'Opacity' images, indicative of potential pneumonia. The low recall in both cases suggests that these models could miss a substantial number of pneumonia cases, which is a significant issue from a healthcare perspective. Further work should focus on improving the recall rate for 'Opacity' images, potentially by gathering more labeled 'Opacity' data, implementing data augmentation techniques, experimenting with different model architectures or using techniques to address class imbalance such as oversampling the minority class or undersampling the majority class.

Comparison of Results

Threshold Impact: The table clearly shows that varying thresholds have significant impacts on the performance metrics. Lower thresholds tend to favor higher recall at the expense of other metrics, while higher thresholds might balance accuracy and precision but reduce recall.

Model Complexity: The CNN models generally have a shorter execution time compared to more complex models like VGG16 and Mask R-CNN.

Dataset Influence: Using all images converted to PNG generally seems to affect performance differently across models. Some models improved, while others did not.

Object Detection vs Classification: Mask R-CNN for object detection exhibits unique behaviors in terms of precision and recall, reflecting the different nature of the task compared to pure classification models.

8. Limitations

What are the limitations of your solution? Where does your model fall short in the real world? What can you do to enhance the solution?

Trade-offs between Recall and Other Metrics: Higher recall might be critical in medical applications to reduce false negatives. However, the increase in recall often comes at the cost of accuracy and precision, indicating a need for careful threshold management.

Effect of Architecture and Training Strategies: The results show that different architectures and training metrics (e.g., training with Recall or with Recall & Accuracy) lead to different behaviors. This emphasizes the importance of model selection and hyperparameter tuning.

Execution Time Considerations: The execution time for different models varies significantly. For real-world applications, this aspect must be considered, especially when the dataset size is large.

Improvements with Image Conversion: The conversion of images to PNG has shown mixed results, indicating that image preprocessing needs to be adapted specifically for each model and task.

Major Limitation is non availability of systematic hyper parameter tuning tool. To train a Mask-R-CNN models require time and processing as well as memory resources. Hence to search for better hyper parameter tuning advanced techniques need to be implemented. But due to paucity of time it could not be done.

9. Closing Reflections

What have you learned from the process? What you do differently next time?

Important notes and Future improvements

The results obtained from various models, thresholds, and image processing methods demonstrate the complexity and multifaceted nature of the problem. Key insights include:

The critical role of threshold management in balancing recall with other performance parameters.

The need to carefully select and tune model architectures based on the specific requirements of the task.

The importance of considering execution time, especially for large-scale applications.

A nuanced understanding of image preprocessing and its effect on different models.

The study offers valuable directions for future work, especially in fine-tuning models, selecting appropriate thresholds, and developing preprocessing techniques tailored to specific tasks. Combining different models or using ensemble methods might also provide a path to further improvements. Overall, the comprehensive exploration carried out sets a strong foundation for advancing the state-of-the-art in medical image analysis for pneumonia detection.

Deployment using Flask

Now we developed a model , That can able to classify the DICOM images Opacity and No Opacity classes and masks or the bounding boxes of the corresponding images, Now we are deploying the model using flask ,for this purpose we need to create some code

The link for the code as follows, By clicking on the below link you how we did deployment process "https://drive.google.com/file/d/10HCCg_S2SjbY9HR1zvvqkAn0WOjASMDW/view?usp=drive_link"

The code as follows

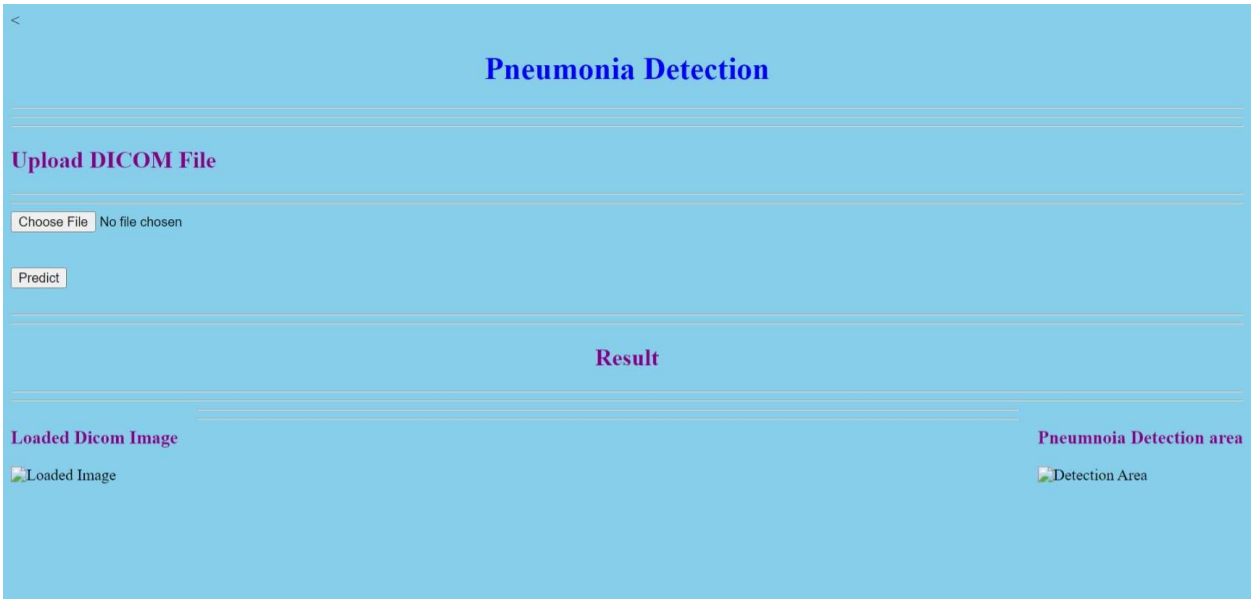
```
from flask import Flask, render_template, request, send_file
import os
import pydicom
from PIL import Image
import numpy as np
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pandas as pf
import tensorflow as tf
from tensorflow.keras.models import load_model
from flask import Flask, render_template, request
IMAGE_HEIGHT = 224
IMAGE_WIDTH = 224

app = Flask(__name__)

def iou_loss(y_true, y_pred):
    y_true = tf.reshape(y_true, [-1])
    y_pred = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true * y_pred)
    score = (intersection + 1.) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) - intersection + 1.)
    return 1 - score

# combine bce loss and iou loss
def iou_bce_loss(y_true, y_pred):
    return 0.5 * keras.losses.binary_crossentropy(y_true, y_pred) + 0.5 * iou_loss(y_true, y_pred)
def mean_iou(y_true, y_pred):
    y_pred = tf.round(y_pred)  # Ensure predictions are binary (0 or 1)

    intersection = tf.reduce_sum(y_true * y_pred, axis=[1, 2])
    union = tf.reduce_sum(y_true + y_pred, axis=[1, 2]) - intersection

    iou = (intersection + tf.keras.backend.epsilon()) / (union + tf.keras.backend.epsilon())

    return tf.reduce_mean(iou)
# Register the custom loss function in the Keras custom_object_scope
custom_objects = {"iou_bce_loss": iou_bce_loss,
                  "mean_iou":mean_iou
}
#place the model
model = load_model("C:/Users/Admin/Downloads/Capstone_Project/Deployment/final_deployment/maskrcnn_model_dcm_21k.h5",custom_objects=custom_objects)
#here we specified model path

# Define the path to store converted PNG images
ORG_INPUT_FOLDER = "original/png_images"
PNG_OUTPUT_FOLDER = "static/png_images"
os.makedirs(PNG_OUTPUT_FOLDER, exist_ok=True)
os.makedirs(ORG_INPUT_FOLDER, exist_ok=True)

def preprocess(dicom_path):
    im = pydicom.dcmread(dicom_path)
    im = im.pixel_array
    im = cv2.resize(im, (224, 224))
    return im

def convert_to_png(pixel_array, output_path):
    # Normalize the pixel array to the range [0, 255]
    pixel_array = (pixel_array - pixel_array.min()) * (255 / (pixel_array.max() - pixel_array.min()))
    pixel_array = pixel_array.astype(np.uint8)

    # Convert the image to RGB mode if it's grayscale
    if len(pixel_array.shape) == 2:
        pixel_array = cv2.cvtColor(pixel_array, cv2.COLOR_GRAY2RGB)

    image = Image.fromarray(pixel_array)
    image.save(output_path)
```

```
@app.route("/", methods=["GET", "POST"])
def upload_dicom():
    if request.method == "POST":
        dicom_file = request.files["file"]
        if dicom_file:
            dicom_path = os.path.join(PNG_OUTPUT_FOLDER, "uploaded.dcm")
            dicom_file.save(dicom_path)
            im = pydicom.dcmread(dicom_path)
            im = im.pixel_array
            im = cv2.resize(im, (IMAGE_HEIGHT, IMAGE_WIDTH))
            im = np.stack((im,) * 3, -1)
            output_path = os.path.join(PNG_OUTPUT_FOLDER, "input.png")
            convert_to_png(im, output_path)
            im = im / 255.0  # Normalize the image to [0, 1]
            image = np.expand_dims(im, axis=0)
            # Perform inference
            predictions = model.predict(image)
            # Threshold the mask to convert probabilities to binary values
            threshold_value = 0.10  # Adjust this threshold as per your needs
            binary_mask = (predictions >= threshold_value).astype(np.uint8)
            # Find contours in the binary mask
            contours, _ = cv2.findContours(binary_mask[0], cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            # Initialize a list to store the bounding boxes (x, y, width, height)
            bounding_boxes = []
            # Iterate through all detected contours and calculate bounding boxes
            for contour in contours:
                x, y, width, height = cv2.boundingRect(contour)
                bounding_boxes.append((x, y, width, height))
            if len(bounding_boxes) == 0.0:
                Pneumonia = "The patient having No Opacity"
                output_path = os.path.join(PNG_OUTPUT_FOLDER, "uploaded.png")
                convert_to_png(im, output_path)
            else:
                Pneumonia = "The patient having Opacity"
                image_1 = preprocess(dicom_path)
                for box in bounding_boxes:
                    x, y, width, height = box
                    x1, y1 = int(x), int(y)
                    x2, y2 = int(x + width), int(y + height)
                    color = (0,255, 0)  # red color for the bounding boxes
                    thickness = 2
                    text = "Opacity"
                    imag_bb = cv2.rectangle(image_1, (x1, y1), (x2, y2), color, thickness)
                    cv2.putText(imag_bb, text, (x1, y1 - 5), cv2.FONT_ITALIC, 0.3,color, 2)
                output_path = os.path.join(PNG_OUTPUT_FOLDER, "uploaded.png")
                convert_to_png(imag_bb, output_path)

            return render_template("upload.html",
                                   Pneumonia=Pneumonia,
                                   image_path1="static/png_images/input.png",
                                   image_path="static/png_images/uploaded.png")

    return render_template("upload.html")

if __name__ == "__main__":
    app.run(debug=True)
```

Here you can see how our flask application API look like

And the HTML link be given as

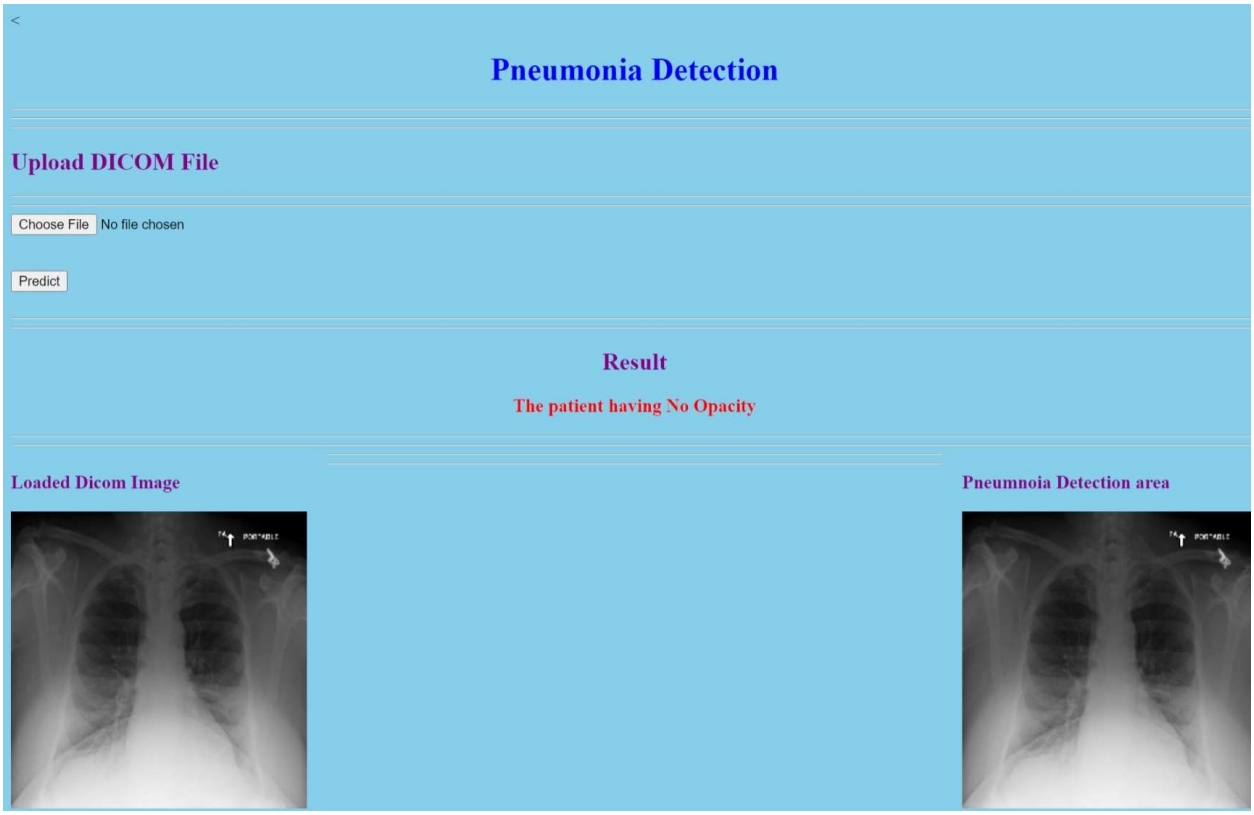
“https://drive.google.com/file/d/1kyC7s749WWV2Hk1cvJ1WGZt4LZD4JhkY/view?usp=drive_link”



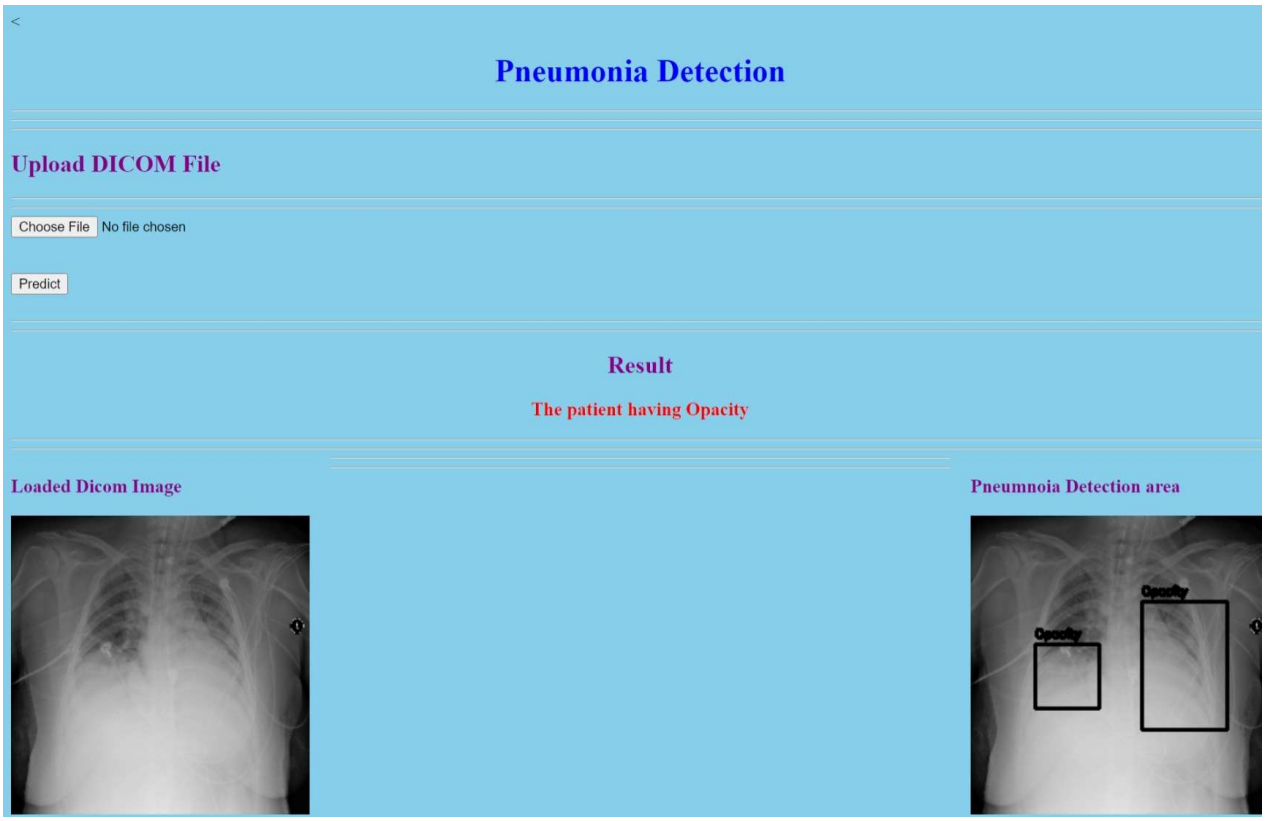
In the above shown ,once you click on choose file Option,it will direct to our drive to choose a dicom image(will not accept any other format), then the image has been uploaded.

Once the image is uploaded , if we click on predict button ,it will predict that the patient having Opacity or Non Opacity and reflects the loaded image and if it is Opacity class it will reflect the area where it was identified with bounding boxes other wise simply reflects the same image.

Here I am showing how it will look like,



The above Image reflects No opacity class



The above Image shown Opacity class and along with the image with bounding boxes

By clicking the given video link, we can able to see how the flask app work like

“https://drive.google.com/file/d/1ANdWyNbctqZlTl7CivagrGPkqglXjjjP/view?usp=drive_link”