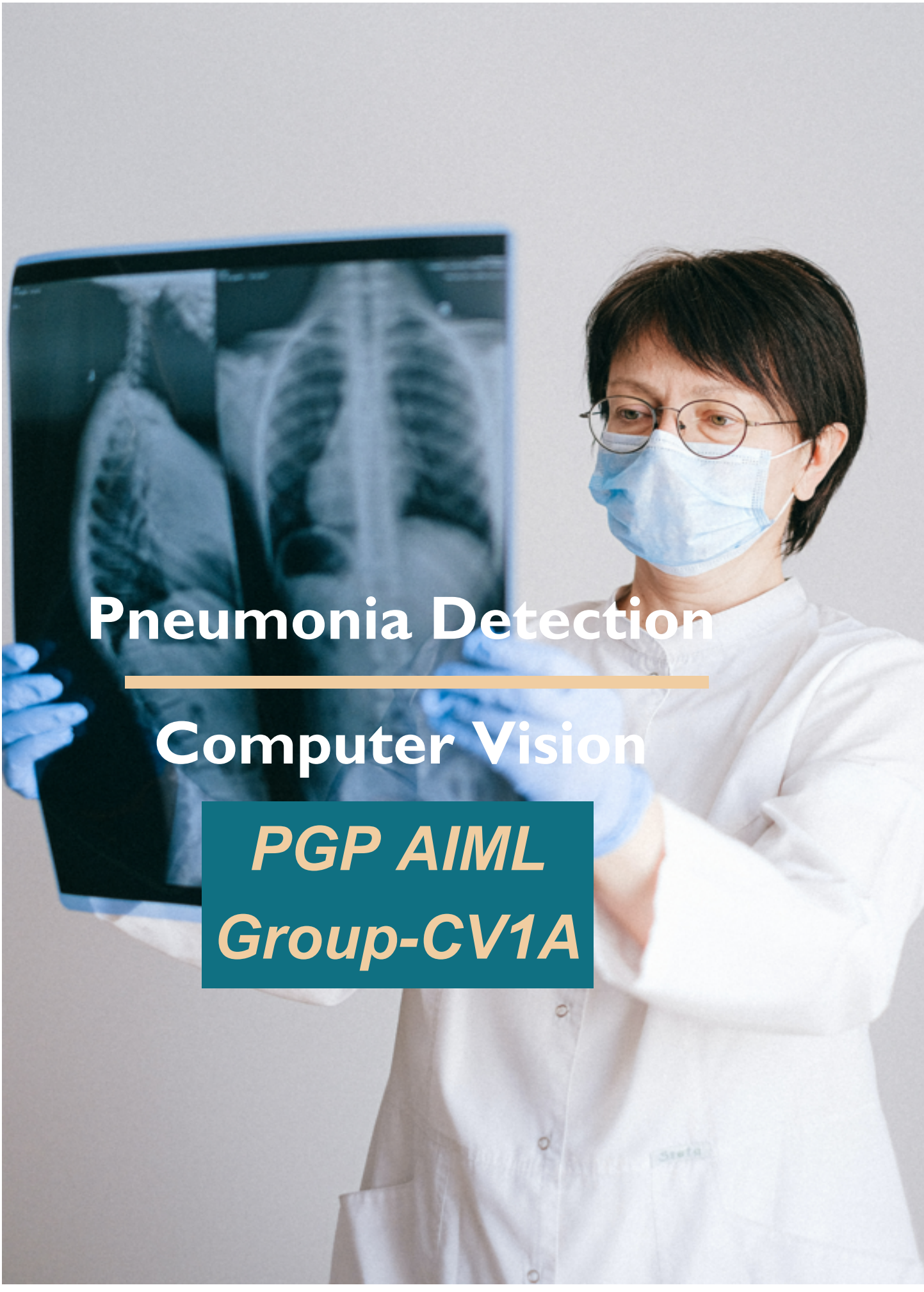


Pneumonia Detection

Computer Vision

PGP AIML
Group-CV1A

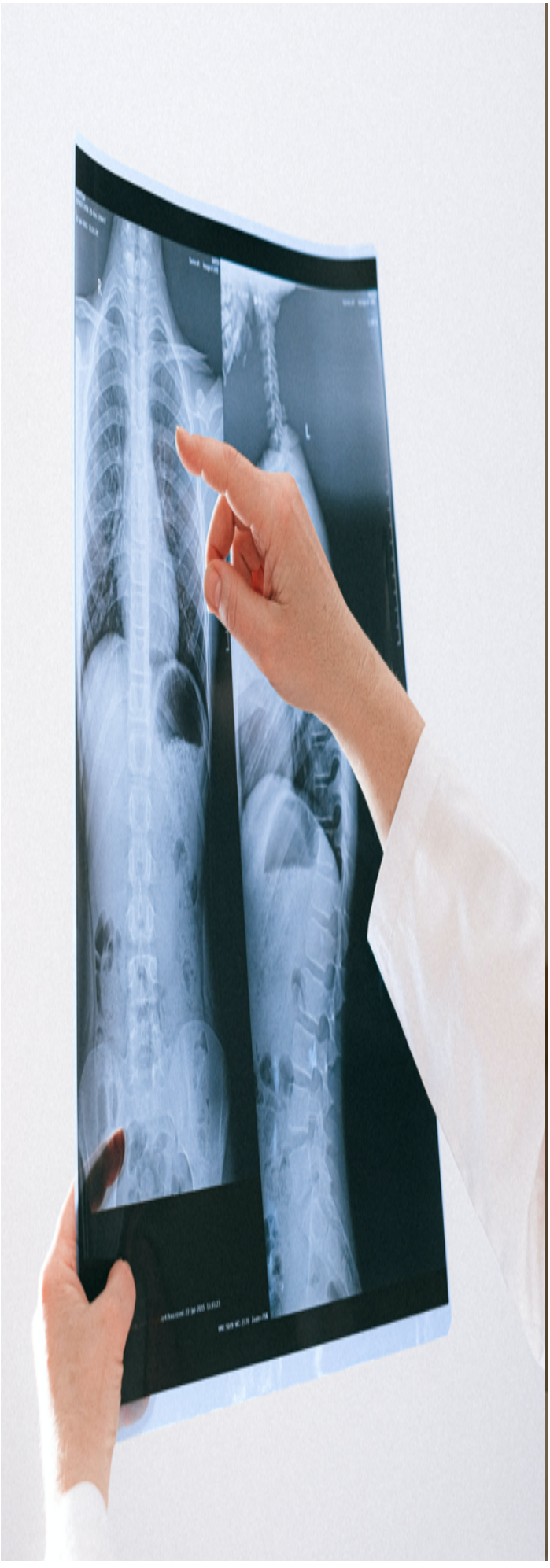


TABLE OF CONTENTS

Capstone Project -AIML PGP

• DOMAIN: Health Care

• CONTEXT: Computer vision can be used in health care for identifying diseases. In Pneumonia detection we need to detect Inflammation of the lungs. In this challenge, you're required to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, your algorithm needs to automatically locate lung opacities on chest radiographs.

• DATA DESCRIPTION: - In the dataset, some of the features are labeled "Not Normal No Lung Opacity". This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia. Dicom original images: - Medical images are stored in a special format called DICOM files (*.dcm). They contain a combination of header metadata as well as underlying raw image arrays for pixel data. - Dataset has been attached along with this project.

Please use the same for this capstone project. - Original link to the dataset : https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data [for your reference only]. You can refer to the details of the dataset in the above link - Acknowledgements: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/overview/acknowledgements.

1. Summary of problem statement, data and findings

Problem Statement:

Deploying Deep Learning techniques and algorithm need to automatically locate lung opacities on chest radiographs.

Abstract:

Pneumonia is a life-threatening infectious disease affecting one or both lungs in human being commonly caused by bacteria. Chest X-Rays which are used to diagnose pneumonia need expert radiotherapists for evaluation. Thus, developing an automatic system for detecting pneumonia would be beneficial for treating the disease without any delay particularly in remote areas. Due to the success of deep learning algorithms in analyzing medical images, Convolutional Neural Networks (CNNs) have gained much attention for disease classification. In addition, features learned by pre-trained CNN models on large-scale datasets are much useful in image classification tasks. In this work, we appraise the functionality of pre-trained CNN models utilized as feature extractors followed by different classifiers for the classification of abnormal and normal chest XRays. We analytically determine the optimal CNN model for the purpose.

Data Set:

The dataset used is ChestX-Ray which consists of 26,684 frontal chest X-ray images from 30,227 patients. Chest X-Ray Images (Pneumonia) dataset of 3.5 GB size has been imported from drive, with total of 29710 jpg images split into Train, Test folders each divided into 3 classes No Lung Opacity / Not Normal, Normal, Lung Opacity. In image dataset multiple entries for the same patientId with different image coordinates. These are different images possibly belonging to one patient. First dataset contains all images and the second dataset contains classes. merging both dataset helps to train the CNN model.

	patientId	x	y	width	height	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1
5	00436515-870c-4b36-a041-de91049b9ab4	562.0	152.0	256.0	453.0	1
6	00569f44-917d-4c86-a842-81832af98c30	NaN	NaN	NaN	NaN	0
7	006cec2e-6ce2-4549-bffa-eadfcd1e9970	NaN	NaN	NaN	NaN	0
8	00704310-78a8-4b38-8475-49f4573b2dbb	323.0	577.0	160.0	104.0	1
9	00704310-78a8-4b38-8475-49f4573b2dbb	695.0	575.0	162.0	137.0	1

```
Train_labels shape : (30227, 6)
Train labels: Unique patientIds we have:  26684
Train labels: pure duplicates (all rows same) :  0
```

stage_2_train_labels

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
5	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
6	00569f44-917d-4c86-a842-81832af98c30	No Lung Opacity / Not Normal
7	006cec2e-6ce2-4549-bffa-eadfcd1e9970	No Lung Opacity / Not Normal
8	00704310-78a8-4b38-8475-49f4573b2dbb	Lung Opacity
9	00704310-78a8-4b38-8475-49f4573b2dbb	Lung Opacity

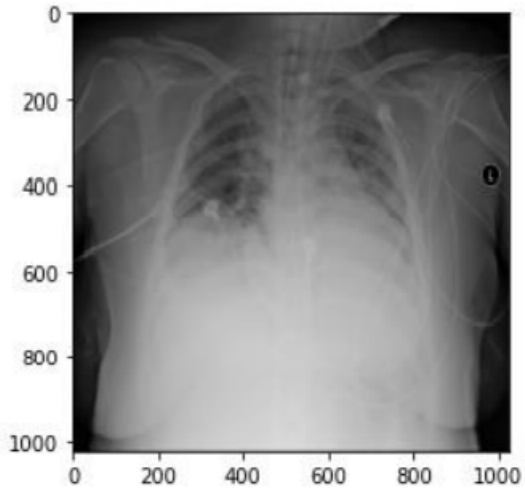
```
Detailed_class_info shape  (30227, 2)
Detailed_class_info: Unique patientIds we have:  26684
Detailed_class_info: pure duplicates (all rows same) :  3543
```

stage_2_detailed_class_info

Findings:

- This project presents CNN models to accurately detect pneumonic lungs from chest X-rays, which can be utilized in world by medical practitioners to treat pneumonia. These models have been trained to classify chest X-ray images into normal and pneumonia in a few seconds, hence serving the purpose of early detection of pneumonia. We see patient ids, and bounding box is present in the dataset. 0 means No Pneumonia, 1 means Pneumonia Bounding Box is not present when the patient does not have pneumonia, however 0 can mean No Lung Opacity/Not Normal.

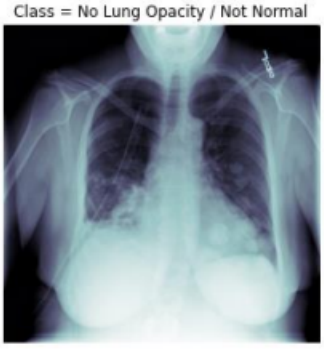
Dicom images along with metadata:



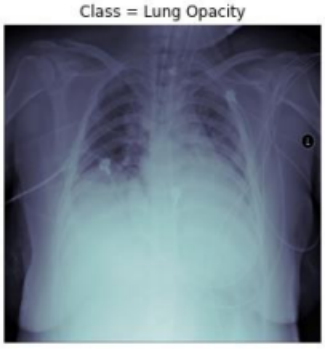
```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 200
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.6379.1517874325.469569
(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: 1.2.276.0.7230010.3.1.4.8323329.6379.1517874325.469569
(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: AP'
(0010, 0010) Patient's Name                      PN: '00436515-870c-4b36-a041-de91049b9ab4'
(0010, 0020) Patient ID                          LO: '00436515-870c-4b36-a041-de91049b9ab4'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '32'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'AP'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.6379.1517874325.469568
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.6379.1517874325.469567
(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: [0.139, 0.139]
```

Class = Normal

Class = No Lung Opacity / Not Normal



Class = Lung Opacity



2. Overview Of the Final Process

The problem we are addressing in this project is to develop a computer vision-based system for detecting pneumonia in chest X-ray images. The goal is to accurately identify whether a given chest X-ray image shows signs of pneumonia or not by placing a bounding box in place of pneumonia.

To train our model, we used a publicly available dataset called the Chest X-Ray Images (Pneumonia) dataset from rsna-pneumonia-detection-challenge. This dataset consists images for the current stage - provided as **stage_2_train_images.zip** and **stage_2_test_images.zip**. we will also need the training data - **stage_2_train_labels.csv** - and the sample submission **stage_2_sample_submission.csv**, which provides the IDs for the test set, as well as a sample of submission should look like. The file **stage_2_detailed_class_info.csv** contains detailed information about the positive and negative classes in the training set.

The training data is provided as a set of patientIds and bounding boxes. Bounding boxes are defined as follows:
x-min y-min width height

There is also a binary target column, Target, indicating pneumonia or non-pneumonia.

There may be multiple rows per patientId.

We are predicting whether pneumonia exists in a given image or not. We do so by predicting bounding boxes around areas of the lung. Samples without bounding boxes are negative and contain no definitive evidence of pneumonia. Samples with bounding boxes indicate evidence of pneumonia.

Dataset Details:

File descriptions

- **stage_2_train.csv** - the training set. Contains patientIds and bounding box / target information.
- **stage_2_sample_submission.csv** - a sample submission file in the correct format. Contains patientIds for the test set. Note that the sample submission contains one box per image, but there is no limit to the number of bounding boxes that can be assigned to a given image.
- **stage_2_detailed_class_info.csv** - provides detailed information about the type of positive or negative class for each image.

Data fields

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

Methodology:

The primary goal of using Convolutional Neural Network in most of the image classification tasks is to reduce the computational complexity of the model which is likely to increase if the input are images. The original 3-channel images were resized from 1024×1024 into 256×256 pixels to reduce the heavy computation and for faster processing. Final dataset used to map training and testing images to its classes. The 'Lung Opacity" class has images with the presence of fuzzy clouds of white in the lungs, associated with pneumonia. The regions of lung opacities are labelled with a bounding box.

patientId are same in both dataframes.
Merged dataframe

	patientId	x	y	width	height	Target	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity
5	00436515-870c-4b36-a041-de91049b9ab4	562.0	152.0	256.0	453.0	1	Lung Opacity
6	00569f44-917d-4c86-a842-81832af98c30	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
7	006cec2e-6ce2-4549-bffa-eadfcd1e9970	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
8	00704310-78a8-4b38-8475-49f4573b2dbb	323.0	577.0	160.0	104.0	1	Lung Opacity
9	00704310-78a8-4b38-8475-49f4573b2dbb	695.0	575.0	162.0	137.0	1	Lung Opacity

(30227, 7)

Merged Dataframe

Image with bounding box

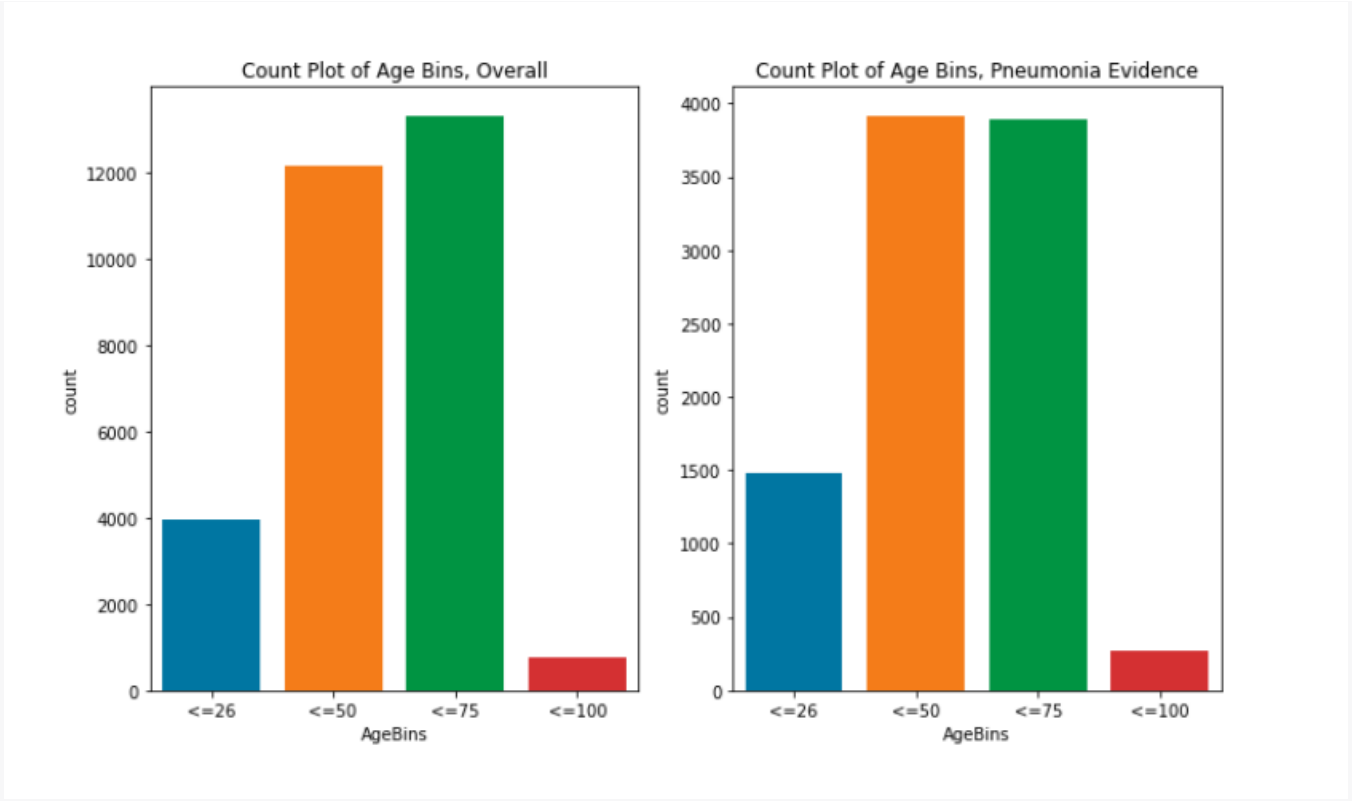
Pre-processing:

In Preprocessing we are cleaning Noisy data by using method Binning. Using Binning Method for PatientAge feature We'll make use of a pd.cut which is 'Bin values into discrete intervals'. Use of this method is recommended when the need is to segment and sort data values into bins. This function is also useful for going from a continuous variable to a categorical variable. Supports binning into an equal number of bins, or a pre-specified array of bins.

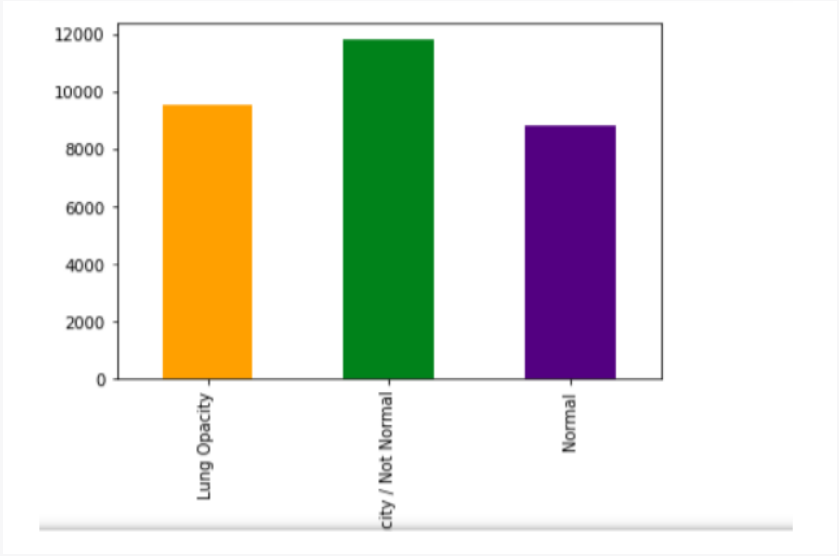
```
Checking outliers in `PatientAge
--------------------------------------------------------------------------------
Minimum `PatientAge` in the training dataset: 1.0
Maximum `PatientAge` in the training dataset: 100.0
75th Percentile of `PatientAge` in the training dataset: 59.0
`PatientAge` in upper whisker for box plot: 84.0

Using pd.clip to set upper threshold of 100 for age and remove outliers
--------------------------------------------------------------------------------
Get the distribution of `PatientAge` overall and where Target = 1
--------------------------------------------------------------------------------
Creating Age Binning field ------------------------------------------------------------
Value counts of the age bin field created
--------------------------------------------------------------------------------
```

	Counts of Age Bins, Overall	Counts of Age Bins, Target=1
<=26	3972	1478
<=50	12157	3917
<=75	13318	3895
<=100	780	265

```
Exploring the bounding boxes centers for `AgeBins` for random sample = 200
```

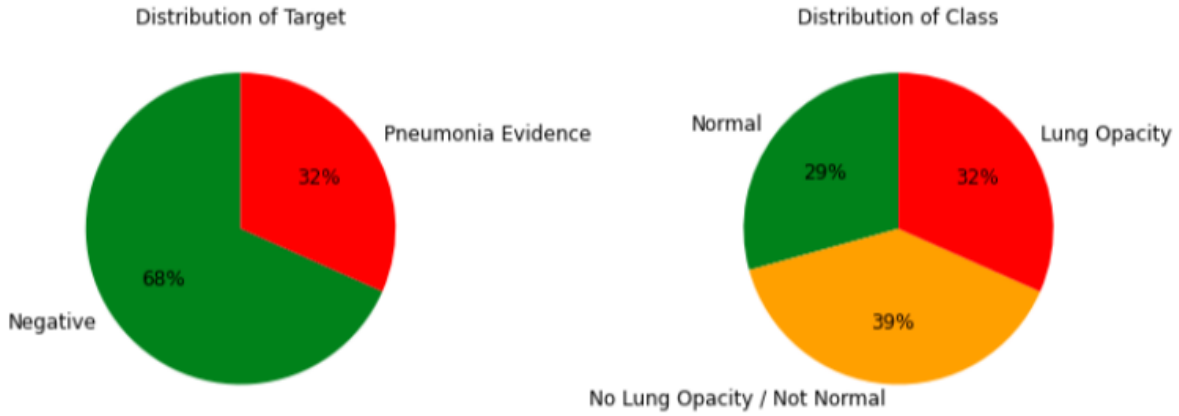


We are using univariate graphical EDA type for data analysis. Multiple or Grouped charts: Grouped bar charts are bar charts representing multiple sets of data items for comparison where a single color is used to denote one specific series in the dataset.



Box Plots:

These are used to display the distribution of quantitative value in the data. If the data set consists of categorical variables, the plots can show the comparison between them.

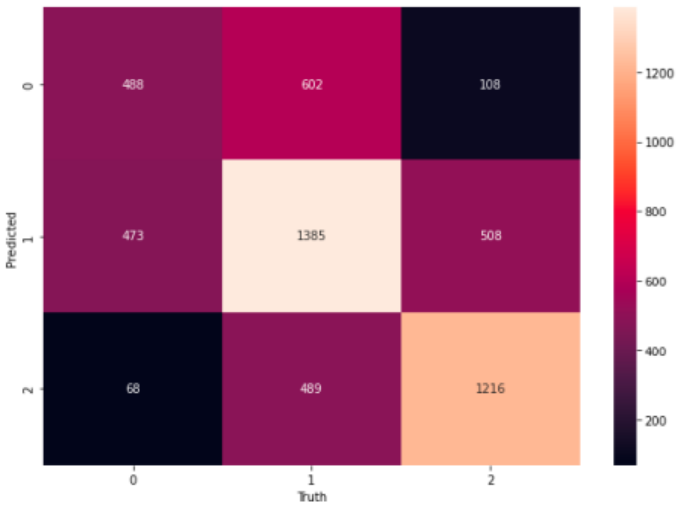


We are using Multivariate graphical data to show the relationships between two or more variables. Here the outcome depends on more than two variables. For this graphical data representation, we used a heat map.

A Heat map is a colored graphical representation of multivariate data structured as a matrix of columns and rows. The heat map below is for the confusion matrix that was for the basic CNN model used for the Classification.

```
668/668 [==============================] - 11s 5ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4814
           1       1.00      1.00      1.00      9455
           2       1.00      1.00      1.00      7078

    accuracy                           1.00     21347
   macro avg       1.00      1.00      1.00     21347
weighted avg       1.00      1.00      1.00     21347
```


```
167/167 [==============================] - 1s 5ms/step
              precision    recall  f1-score   support

           0       0.47      0.41      0.44      1198
           1       0.56      0.59      0.57      2366
           2       0.66      0.69      0.67      1773

    accuracy                           0.58      5337
   macro avg       0.57      0.56      0.56      5337
weighted avg       0.57      0.58      0.58      5337
```



Exploratory Data Analysis (EDA) is one of the techniques used for extracting vital features and trends used by machine learning and deep learning models in Data Science. Thus, EDA has become an important milestone for anyone working in data science. There are some steps involved in EDA:

We have Data collected from huge volumes and various forms of relevant data for analysis, which belongs to Pneumonia Detection. Find all variable like class and labels from collected data.

In this step we Cleaning the dataset by removing null values and duplicate records as shown in fig.

```
Count total NaN at each column in the dataset :

 patientId    0
x             0
y             0
width         0
height        0
Target        0
class         0
dtype: int64
Dataframe Info:
```

Observations during EDA:

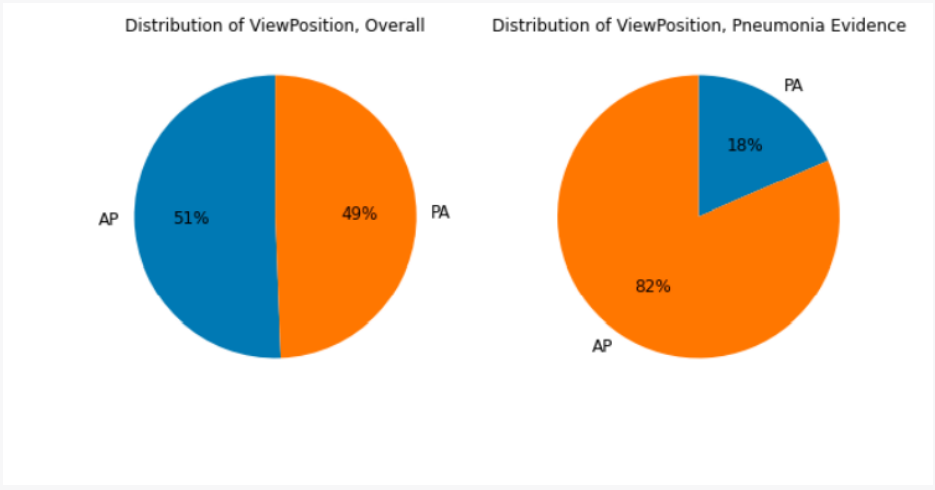
Anterior and Posterior positions:

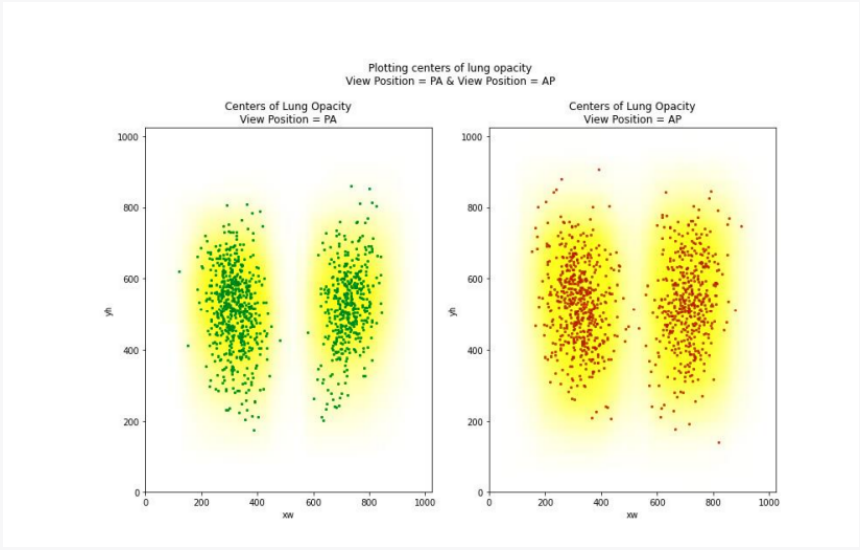
Anterior (AP):

At times, it is not possible for radiographers to acquire a PA chest X-ray. This is usually because the patient is too unwell to stand. So, AP projection images are of lower quality than PA images.

Posterior (PA):

In PA, X-Ray beam hits the posterior (back) part of the chest before the anterior (front) part. While obtaining the image, patient is asked to stand with their chest against the film.





The problem methodology involves using a deep learning model to detect pneumonia from chest X-ray images. The dataset consists of X-ray images labeled as normal or with pneumonia. The salient features of the data include the size and resolution of the images, as well as the presence or absence of opacity in the lung fields.

The first step in the process was to preprocess the data by resizing the images and normalizing the pixel values. The images were then split into training and validation sets, and data augmentation techniques such as rotation, shifting, and flipping were applied to increase the size of the training set.

The deep learning model used in this process was a convolutional neural network (CNN). The CNN architecture consists of multiple layers of convolution, pooling, and dropout, followed by fully connected layers. The model was trained using the Adam optimizer and binary

cross-entropy loss function. The model was evaluated on the validation set, and the hyperparameters were tuned to achieve the best performance.

To combine techniques, transfer learning was used to leverage the pre-trained weights of a ResNet50 model. The pre-trained weights were used to initialize the CNN layers, and the model was fine-tuned on the pneumonia detection task. This approach helped to improve the performance of the model by reducing the training time and achieving higher accuracy.

Once the model was trained, it was saved as a pickle file for future use. A GUI was developed using the tkinter library to enable the user to browse and select an X-ray image for testing. The selected image was preprocessed, and the model was used to predict whether the image had pneumonia or not. The output was displayed in the GUI along with the input image.

3. Step By Step Walkthrough of Solution

Importing the data

Below code is importing the necessary libraries, mounting the Google Drive, and reading in two CSV files: stage_2_train_labels.csv and stage_2_details_class_info.csv.

After reading in the CSV files, the code is checking the shape and the number of unique patient IDs in each dataframe. It also checks for any duplicate entries in the dataframes.

The code then attempts to merge the two dataframes, but realizes that merging the dataframes will result in gaps in the index sequence. Instead, it checks whether the patient IDs are the same in both dataframes, and if they are, it adds the "class" column from the second dataframe to the first dataframe.

Finally, the merged dataframe is displayed and its shape is printed.

Step 1: Import the data.

```
#Mount the Google drive
from google.colab import drive
drive.mount('/content/drive')

#when downloading Project data to Mac, the zip got copied as a folder. Nothing further needed to import data.
images_path = "/content/drive/MyDrive/Colab Notebooks/Capstone - Pneumonia/10. Capstone - Pnemonia"

#Setup the csv files into dataframes - need to read only stage_2_train_labels.csv and stage_2_details_class_info.csv
df1A = pd.read_csv(images_path + "/stage_2_train_labels.csv")
display(df1A.head(10))
print("Train_labels shape :", df1A.shape)
print("Train labels: Unique patientIds we have: ", df1A['patientId'].nunique())
tmp = df1A[df1A.duplicated(['patientId', 'x', 'y', 'width', 'height', 'Target'])]
print("Train labels: pure duplicates (all rows same) : ", tmp.shape[0])
#there are multiple entries for the same patientId with different image coordinates. these are diff. images possibly, so keep them
#eg. df1A[df1A['patientId'] == '00704310-78a8-4b38-8475-49f4573b2dbb'] there are two different entries.

df1B = pd.read_csv(images_path + "/stage_2_detailed_class_info.csv")
display(df1B.head(10))
print("Detailed_class_info shape ", df1B.shape)
print("Detailed_class_info: Unique patientIds we have: ", df1B['patientId'].nunique())
tmp = df1B[df1B.duplicated(['patientId', 'class'])]
print("Detailed_class_info: pure duplicates (all rows same) : ", tmp.shape[0])
#here though there seem to be duplicates, but they are not. When matched to df1A, they will be separate entries
#eg. df1B[df1B['patientId'] == '00704310-78a8-4b38-8475-49f4573b2dbb']

#do below to merge - but this is punching holes in the index sequence
# df1 = df1A.merge(df1B, on='patientId')
# df1 = df1.drop_duplicates()
# display(df1.head(10))
# print("merged DF shape ", df1.shape)

if (df1A['patientId'] == df1B['patientId']).sum() == df1A.shape[0]:
  print("patientId are same in both dataframes.")
  #Include the class column into the first dataframe
  df1A['class'] = df1B['class']

print("Merged dataframe")
display(df1A.head(10))
df1A.shape
```

Mapping training and test images to its classes.

Hashed code is for pre-processing the training data and creating pickle using hashed code which will run only once, “code executed once, it will create pickle on drive to avoid high execution time parsing all the image files”.

The pullSexAge function adds age and sex as input features to the dataset, and the X and Y lists are populated with the image data and target labels, respectively. The image data is resized to (256, 256) and expanded to have a single channel, and the target variable is label encoded using LabelEncoder() from scikit-learn. Finally, the target variable is one-hot encoded using get_dummies().

The preprocessed training data is stored in the X and Y dash variables, where X is a numpy array of the resized and expanded image data, and Y dash is a pandas dataframe of the one-hot encoded target labels.

```
['Lung Opacity', 'No Lung Opacity / Not Normal', 'Normal']
Train img shape  (26684, 256, 256, 1)
Train target shape  (26684, 3)
```

	0	1	2
0	0	0	1
1	0	1	0
2	0	0	1
3	0	0	1
4	0	0	1

```
0  1  2
0  1  0    11821
   0  1     8851
1  0  0     6012
dtype: int64
```

Next we are loading the test images, preparing them for prediction, and checking for any duplicates with the training set.

Firstly, code reads the sample_submission.csv file to get a list of test images. Then, it loads each test image from the 'stage_2_test_images' folder using the DICOM (Digital Imaging and Communications in Medicine) format and resizes them to 256x256. It checks if the image is present in the sample submission CSV file and labels the image as 'Lung Opacity' if it is present. If not present, it labels the image as 'Unknown'. The labels are not used for classification in this code.

The image data and file names are stored in the X_t and fn_t lists, respectively. These lists are then pickled and saved to disk to avoid time-consuming image loading in the future. The code later unpickles these lists to obtain the test image data and filenames.

After loading the test images, it expands the dimensions of the image array X_t to include a single channel as the model expects a 3D array with three color channels.

A DataFrame, Y_dash_t, is created to store the target values, which are one-hot encoded. However, the code comment notes that this one-hot encoding is not correct, and it should be ignored.

Finally, the code checks for any duplicates between the test image filenames and the training set filenames. If a filename is found to be in both sets, it is counted in the variable 'ext_cnt'.

	patientId	PredictionString
0	0000a175-0e68-4ca4-b1af-167204a7e0bc	0.5 0 0 100 100
1	0005d3cc-3c3f-40b9-93c3-46231c3eb813	0.5 0 0 100 100
2	000686d7-f4fc-448d-97a0-44fa9c5d3aa6	0.5 0 0 100 100
3	000e3a7d-c0ca-4349-bb26-5af2d8993c3d	0.5 0 0 100 100
4	00100a24-854d-423d-a092-edcf6179e061	0.5 0 0 100 100
5	0015597f-2d69-4bc7-b642-5b5e01534676	0.5 0 0 100 100
6	001b0c51-c7b3-45c1-9c17-fa7594cab96e	0.5 0 0 100 100
7	0022bb50-bf6c-4185-843e-403a9cc1ea80	0.5 0 0 100 100
8	00271e8e-aea8-4f0a-8a34-3025831f1079	0.5 0 0 100 100
9	0028450f-5b8e-4695-9416-8340b6f686b0	0.5 0 0 100 100

```
(3000, 2)
Test img shape  (3026, 256, 256, 1)
Test target shape  (3026, 2)
```

	filename
0	257f3de7-2f8f-4dba-a0e7-37fc3e6cc018.dcm
1	252437ed-a4e2-4743-995c-65a4bb133996.dcm
2	268116a6-2304-4316-b5b3-3073fc5467b1.dcm
3	264607e7-a410-4519-b0df-6e5db64c373e.dcm
4	2664366f-4f04-49e1-ab20-19b9173f23bc.dcm

```
No. of test image filenames that are available in training data folder  0
```

Map training and testing images to its annotations.

We are doing train data annotations mapped in this step, and the code is splitting it into train and test sets using the `train_test_split` function from scikit-learn library. The images are then converted to RGB format and the shapes of the train and test data are printed.

```
#Test annotations are unclear - all images have same bounding box.. and all have pnemonia?

#train data, already has the annotations mapped
#split the data to test and training sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_dash, test_size=0.2, random_state=42)
X_train_rgb = np.repeat(X_train[..., np.newaxis], 3, -1)
X_train_rgb = np.squeeze(X_train_rgb)
X_test_rgb = np.repeat(X_test[..., np.newaxis], 3, -1)
X_test_rgb = np.squeeze(X_test_rgb)
print(X_train_rgb.shape)
print(X_test_rgb.shape)
```

```
(21347, 256, 256, 3)
(5337, 256, 256, 3)
```

Preprocessing and Visualisation of different classes

In this step we are performing some initial EDA and preprocessing for the Chest X-Ray dataset.

The code first displays the first few rows of two dataframes (df1A and df1B) containing patient and bounding box information. It then prints the total number of patients and the distinct classes in the dataset, identifies duplicate entries, and creates a bar plot showing the distribution of the classes. Missing values in the bounding box columns are replaced with 0. The code then prints out some summary statistics and information about the dataframes. Finally, it creates a pie chart for the distribution of the target and class.

	patientId	x	y	width	height	Target	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	0.0	0.0	0.0	0.0	0	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity

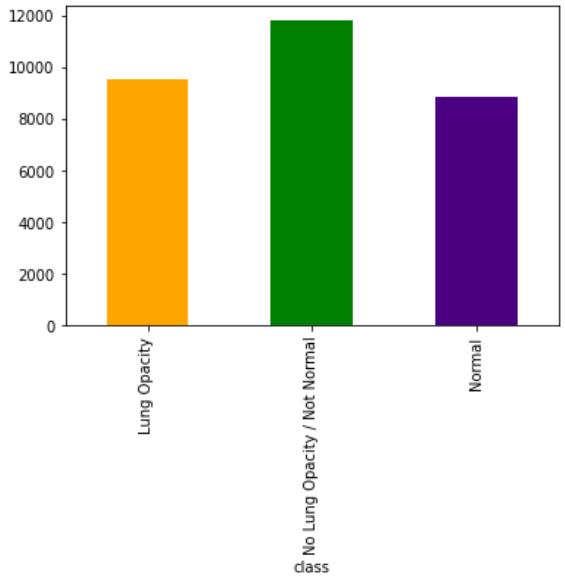
```
Total No of Patients in Class Info 26684
Total distinct classes:  ['No Lung Opacity / Not Normal' 'Normal' 'Lung Opacity']
Duplicate entries (with more than 1 BB)  3543

Updated data samples:
```

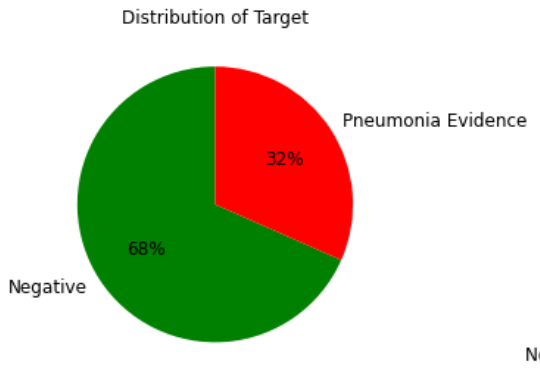
	patientId	x	y	width	height	Target	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	0.0	0.0	0.0	0.0	0	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity

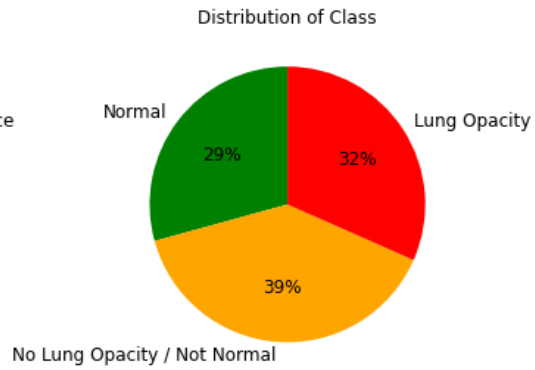
```
Count total NaN at each column in the dataset :

 patientId    0
x             0
y             0
width         0
height        0
Target        0
class         0
dtype: int64
Dataframe Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30227 entries, 0 to 30226
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   patientId  30227 non-null  object
 1   x          30227 non-null  float64
 2   y          30227 non-null  float64
 3   width      30227 non-null  float64
 4   height     30227 non-null  float64
 5   Target     30227 non-null  int64
 6   class      30227 non-null  object
dtypes: float64(4), int64(1), object(2)
memory usage: 1.6+ MB
None
Dataframe describe (numeric data):
```

	x	y	width	height	Target
count	30227.000000	30227.000000	30227.000000	30227.000000	30227.000000
mean	124.561683	115.960962	69.060575	104.084825	0.316108
std	216.326397	190.012883	106.910496	176.932152	0.464963
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	193.000000	231.000000	169.000000	188.000000	1.000000
max	835.000000	881.000000	528.000000	942.000000	1.000000

Lets check the distribution of `Target` and `class` column







The code then creates a dataframe containing the number of bounding boxes associated with each patient, and then prints the number of patients associated with each number of bounding boxes; it then confirms that each patient is associated with only one class. The code merges the two dataframes, and defines a function to visualize a single patient's x-ray image and bounding box information. Finally, the function is used to display the x-ray and bounding box information for one patient.

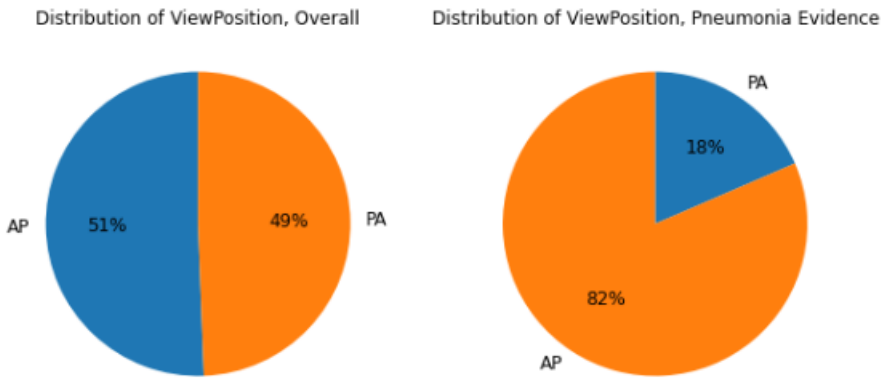
```
Let's also check whether each patientId has only one type of class
--------------------------------------------------------------------------------
Yes, each patientId is associated with only 1 class
Shape of the dataset after the merge: (30227, 8)
Patient Id:  00436515-870c-4b36-a041-de91049b9ab4

Bounding Box Coordinates, X:  264.0

Bounding Box Coordinates, Y:  152.0

Bounding Box Coordinates, Width:  213.0

Bounding Box Coordinates, Height:  379.0
Dataset file meta ------------------------------
```

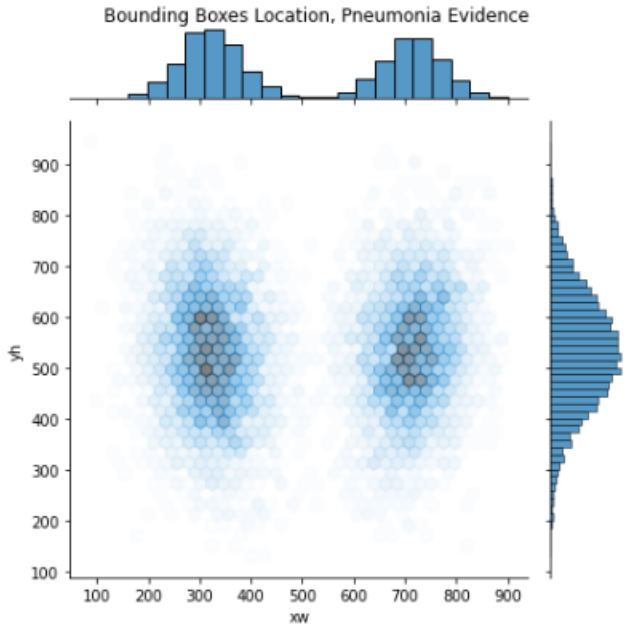
Next code reads DICOM images from directory, extracts additional features from the images, merges the extracted features with the existing **`train_class_df`** dataframe, and displays some information about the resulting dataframe.

```
Read the training images file names and path
--------------------------------------------------------------------------------
Number of images in the training folder: 26811
Columns in the training images dataframe: ['path', 'patientId']
Merge path from the `images` dataframe with `train_class` dataframe
--------------------------------------------------------------------------------
Shape of the `train_class` dataframe after merge: (30227, 15)
(30227, 15)
As expected unique in `BodyPartExamined` is: CHEST
Unique in `Modality` is: CR
--------------------------------------------------------------------------------
```

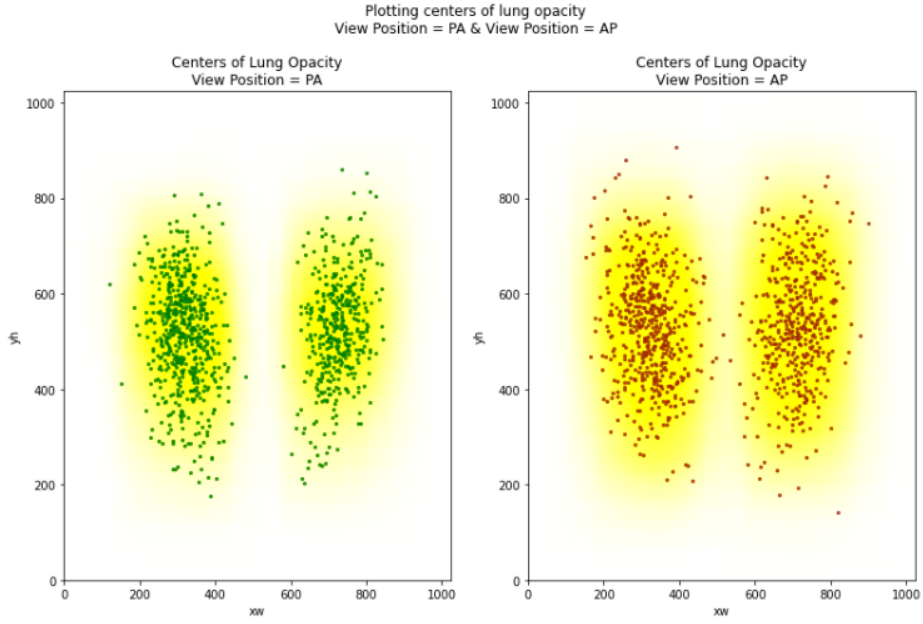
The code generates exploratory data analysis (EDA) plots and summary statistics to examine the distribution of various features in the RSNA Pneumonia Detection Challenge dataset, including ViewPosition, bounding boxes location, PatientAge, and Target, with a focus on identifying potential patterns, outliers, and biases in the data.

```
Overall the distribution is almost equal for `ViewPosition` but where there's a Pneumonia Evidence, `ViewPosition` is `AP`
AP: Anterior/Posterior, PA: Posterior/Anterior
--------------------------------------------------------------------------------
Plot x and y centers of bounding boxes
--------------------------------------------------------------------------------
```





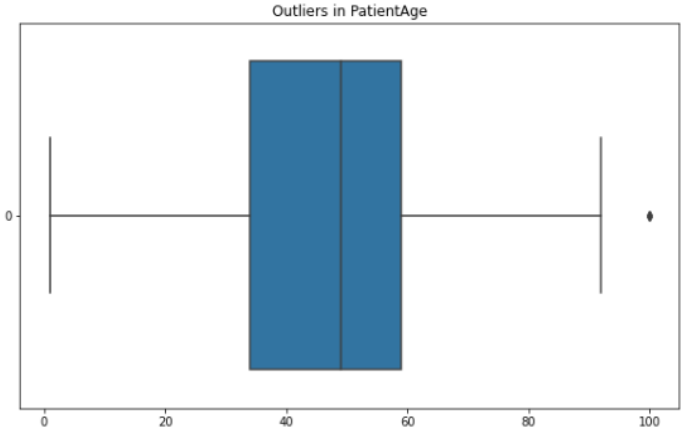
Exploring the bounding boxes centers for `ViewPositions` for random sample = 1000

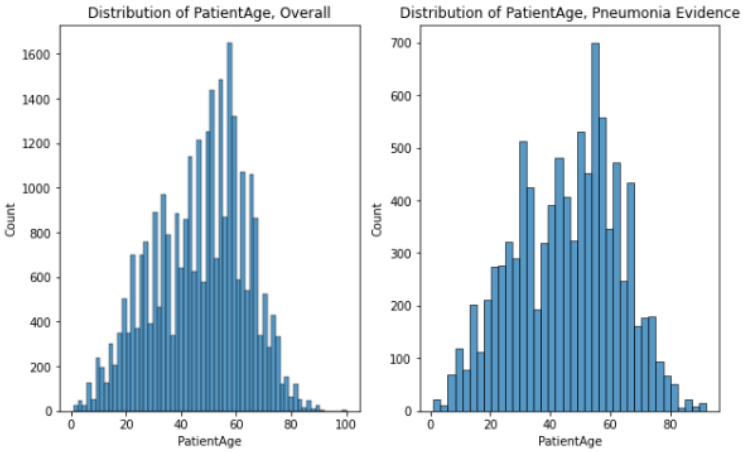


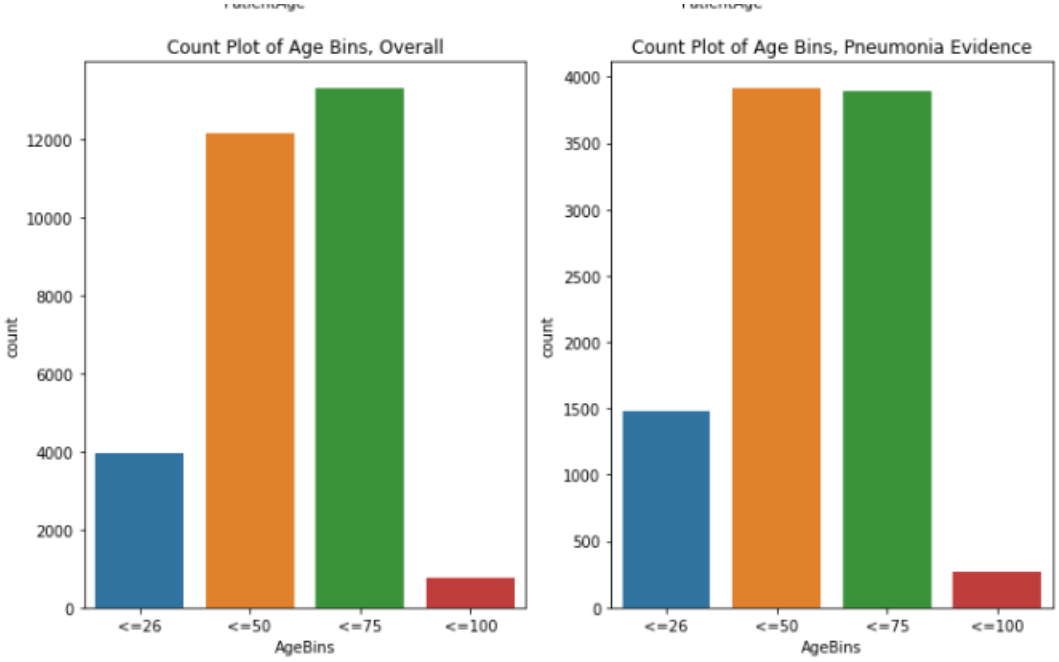
```
Checking outliers in `PatientAge
--------------------------------------------------------------------------------
Minimum `PatientAge` in the training dataset: 1.0
Maximum `PatientAge` in the training dataset: 100.0
75th Percentile of `PatientAge` in the training dataset: 59.0
`PatientAge` in upper whisker for box plot: 84.0

Using pd.clip to set upper threshold of 100 for age and remove outliers
--------------------------------------------------------------------------------
Get the distribution of `PatientAge` overall and where Target = 1
--------------------------------------------------------------------------------
Creating Age Binning field ----------------------------------------------------------------------------------------
Value counts of the age bin field created
--------------------------------------------------------------------------------
```

	Counts of Age Bins, Overall	Counts of Age Bins, Target=1
<=26	3972	1478
<=50	12157	3917
<=75	13318	3895
<=100	780	265

```
Exploring the bounding boxes centers for `AgeBins` for random sample = 200
```





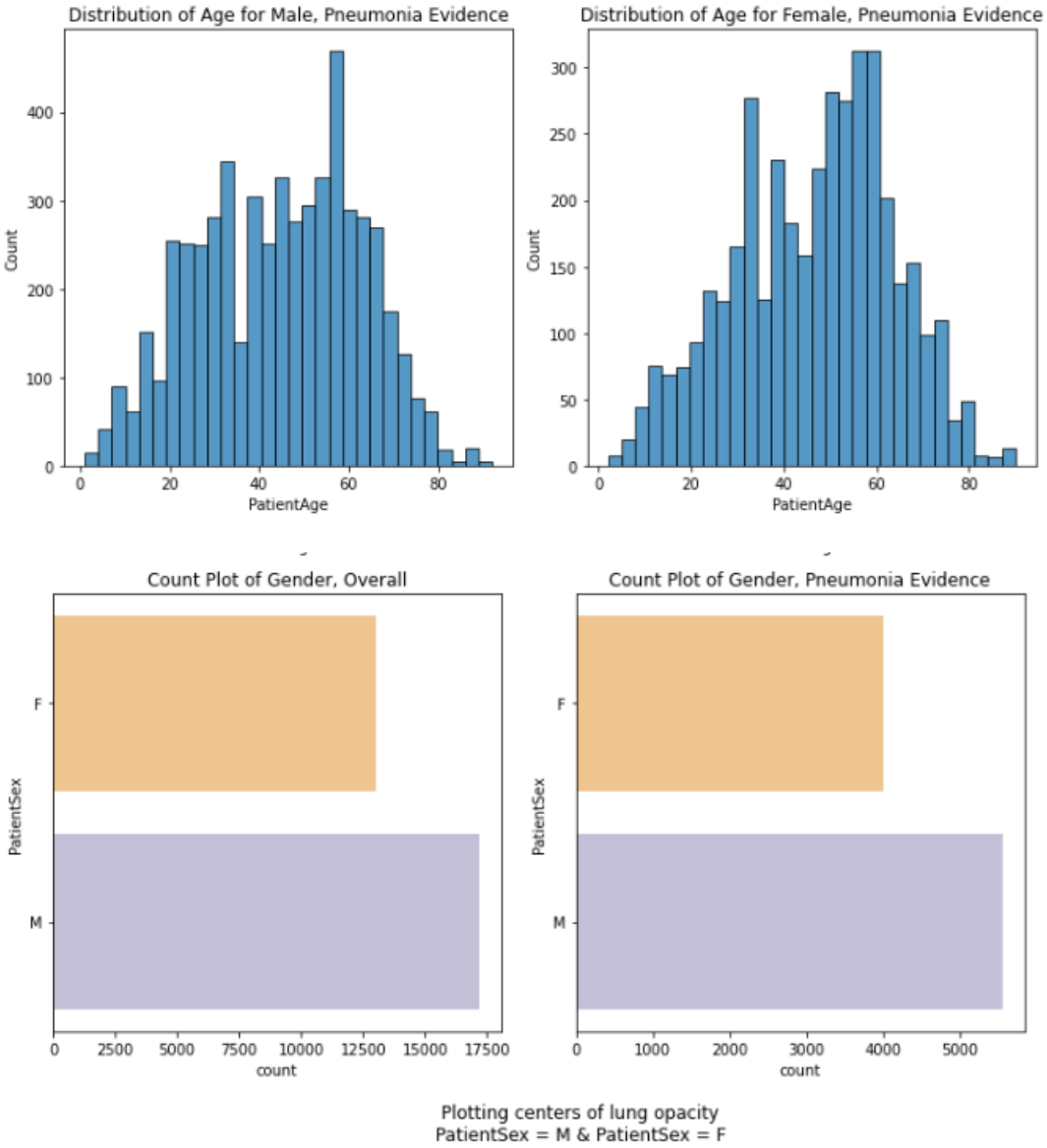


Plotting centers of lung opacity
1 < AgeBins < 26 (Lower Bin) & 76 < AgeBins < 100 (Upper Bin)

```
Checking distribution of age for those with Pneumonia Evidence, by Gender & Count Plot of Gender
--------------------------------------------------------------------------------
```

	% Gender, Overall	% Gender, Target=1
F	0.43	0.42
M	0.57	0.58

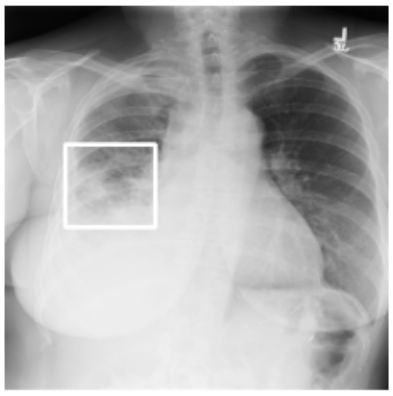
```
Exploring the bounding boxes centers for `PatientSex` for random sample = 1000
```



Display images with bounding box

Here we define two helper functions, **`img_box_coordinates_from_idx`** and **`img_box_coordinates`**, and using them to draw a rectangle on a random image from a dataset, based on the box coordinates of lung opacity obtained from a dataframe.

	0	1	2
10	1	0	0
21	1	0	0
24	1	0	0
28	1	0	0
29	1	0	0

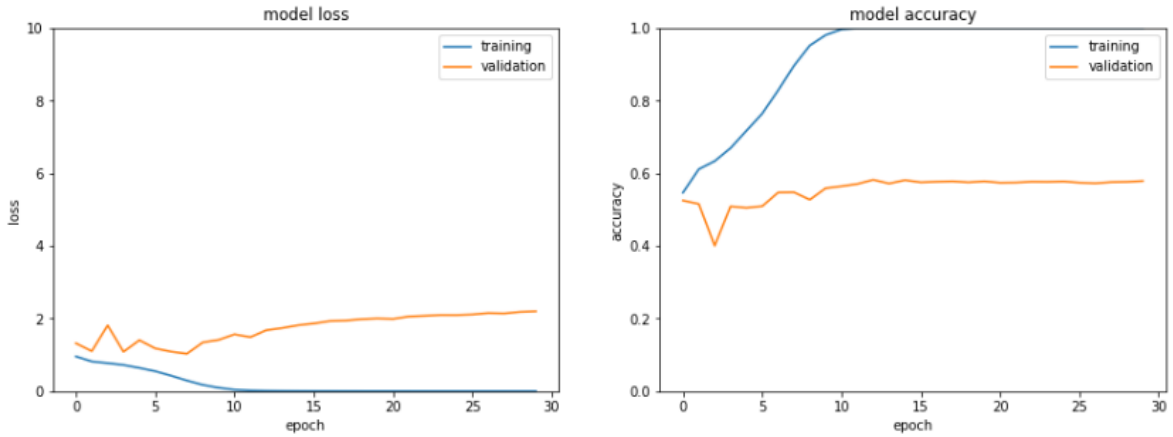


Design, train and test basic CNN models for classification.

In this step we are defining a basic CNN model called "basic_CNN" with the LeNet-5 architecture and compiling it with the Adam optimizer using a learning rate of 0.001, categorical cross-entropy loss, and accuracy as a metric. The model is defined to classify images without considering bounding boxes.

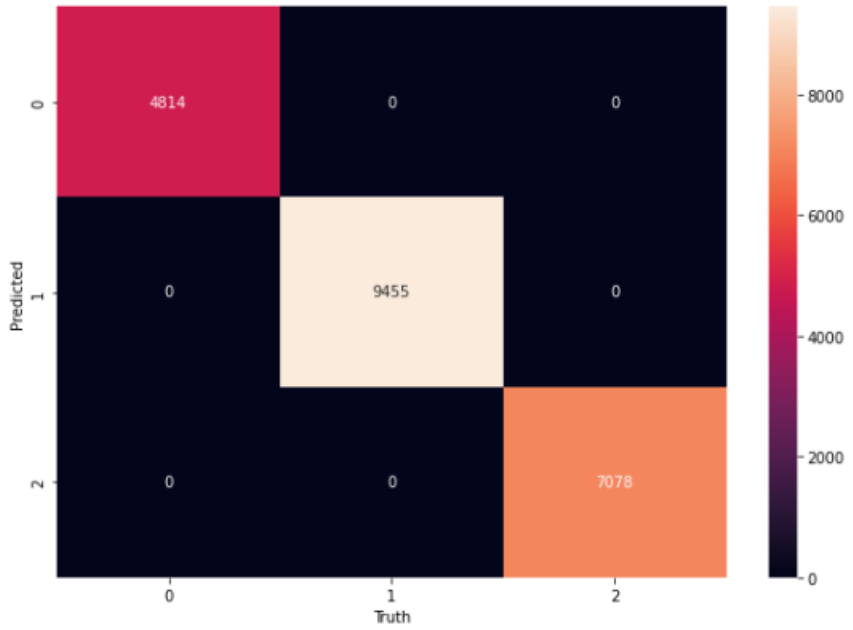
```
X train shape  (21347, 256, 256, 1)
Y train shape  (21347, 3)
X test shape  (5337, 256, 256, 1)
Y test shape  (5337, 3)
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 252, 252, 6)       156

 average_pooling2d (AverageP  (None, 126, 126, 6)      0
 ooling2D)

 conv2d_1 (Conv2D)            (None, 122, 122, 16)      2416

 average_pooling2d_1 (Averag  (None, 61, 61, 16)       0
 ePooling2D)

 flatten_1 (Flatten)          (None, 59536)             0

 dense_2 (Dense)              (None, 120)               7144440

 batch_normalization_1 (Batc  (None, 120)              480
 hNormalization)

 dense_3 (Dense)              (None, 84)                10164

 batch_normalization_2 (Batc  (None, 84)               336
 hNormalization)

 dense_4 (Dense)              (None, 3)                 255

=================================================================
Total params: 7,158,247
Trainable params: 7,157,839
Non-trainable params: 408
_________________________________________________________________
Model compiled!
```

Here we have trained a basic LeNet-5 CNN model for classification without using bounding boxes. This model was trained for 30 epochs on the X_train and Y_train data and validated on the X_test and Y_test data. The training and validation loss as well as accuracy are being plotted and visualized using matplotlib.

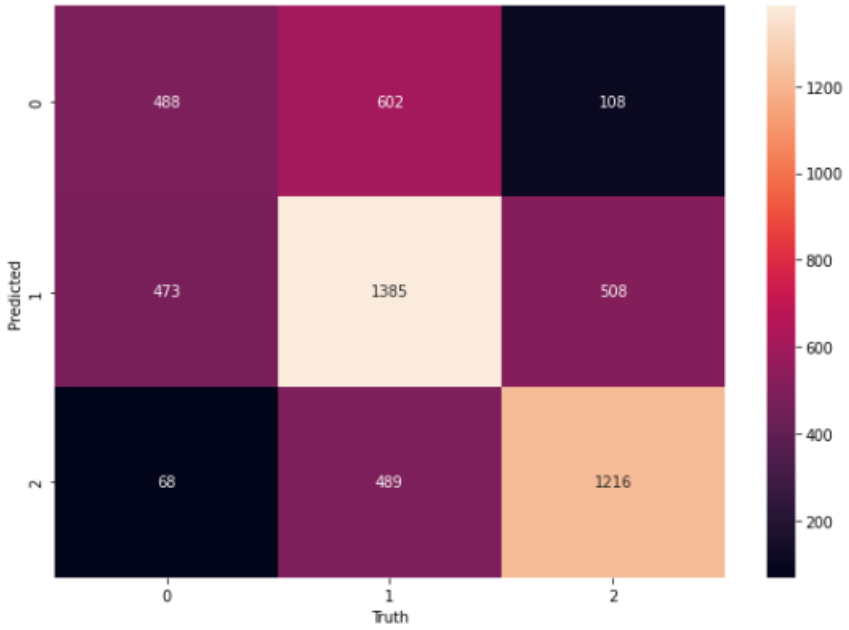


Further we have evaluated the trained basic CNN model for the classification task.. The performance metrics and confusion matrices for both training and testing data are being generated.

```
668/668 [==============================] - 11s 5ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4814
           1       1.00      1.00      1.00      9455
           2       1.00      1.00      1.00      7078

    accuracy                           1.00     21347
   macro avg       1.00      1.00      1.00     21347
weighted avg       1.00      1.00      1.00     21347
```



```
167/167 [==============================] - 1s 5ms/step
              precision    recall  f1-score   support

           0       0.47      0.41      0.44      1198
           1       0.56      0.59      0.57      2366
           2       0.66      0.69      0.67      1773

    accuracy                           0.58      5337
   macro avg       0.57      0.56      0.56      5337
weighted avg       0.57      0.58      0.58      5337
```



Fine tune the trained basic CNN models for classification.

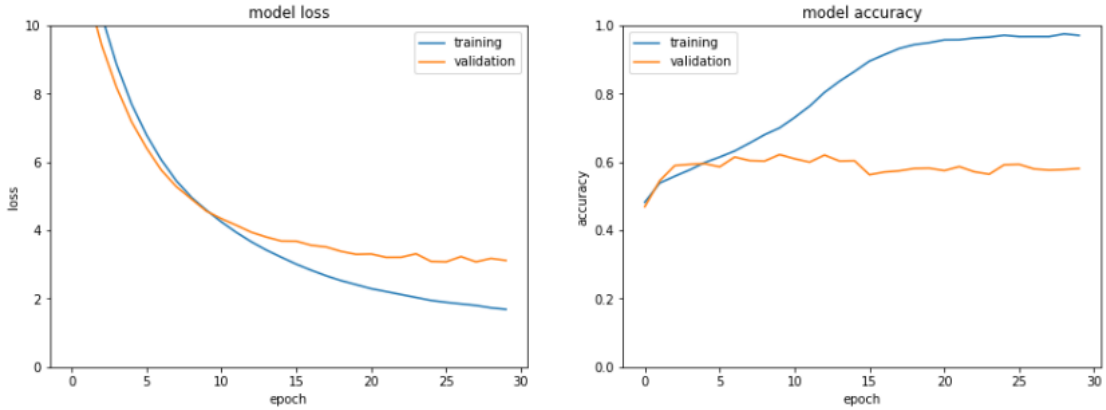
Here in this step, a convolutional neural network (CNN) architecture for image classification with three convolutional layers followed by two fully connected layers is created. The CNN is designed to take input images of size 256x256 with a single channel (grayscale images).

The tuned_basic_CNN function defines the architecture using the Sequential model from Keras. The first layer is a convolutional layer with 8 filters of size 3x3, followed by batch normalization and max pooling. The second convolutional layer has 32 filters of size 3x3, followed by batch normalization and max pooling. The third convolutional layer has 128 filters of size 3x3, followed by batch normalization and max pooling. The fourth and final convolutional layer has 256 filters of size 3x3, followed by batch normalization and max pooling. The output of the fourth convolutional layer is flattened and passed through two fully connected layers of size 256 and 32, respectively. Each fully connected layer is followed by

batch normalization and dropout regularization with a dropout rate of 0.4. Finally, the output layer has 3 units with softmax activation for multiclass classification.

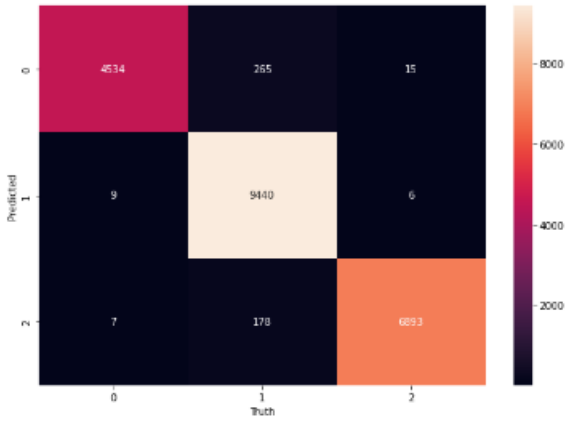
The compile method is called on the model with the Adam optimizer with a learning rate of 0.0001, categorical crossentropy loss, and accuracy as the evaluation metric. This method prepares the model for training.

The fit_generator method is used to train the model. It uses an instance of the ImageDataGenerator class to augment the training data by randomly rotating, zooming, and shifting the images horizontally and vertically. It also randomly flips images horizontally. The flow method of the ImageDataGenerator instance is used to generate batches of augmented data. The fit_generator method takes these batches as input and trains the model for a specified number of epochs.

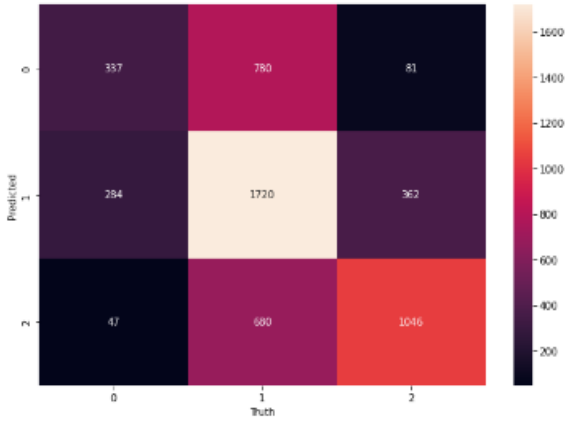


Now , we are evaluating trained model and calculating performance metrics for both the training and testing datasets. The code loads a pickled tuned CNN model, predicts on the training and test data, calculates accuracy metrics and classification reports, generates confusion matrices, and stores the performance results in an Excel file.

```
668/668 [==============================] - 7s 10ms/step
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      4550
           1       1.00      0.96      0.98      9883
           2       0.97      1.00      0.99      6914

    accuracy                           0.98     21347
   macro avg       0.97      0.98      0.98     21347
weighted avg       0.98      0.98      0.98     21347
```




```
167/167 [==============================] - 2s 12ms/step
              precision    recall  f1-score   support

           0       0.28      0.50      0.36       668
           1       0.73      0.54      0.62      3180
           2       0.59      0.70      0.64      1489

    accuracy                           0.58      5337
   macro avg       0.53      0.58      0.54      5337
weighted avg       0.63      0.58      0.59      5337
```



Apply Transfer Learning model for classification

In this step, the code implements transfer learning using the ResNet50 model in Keras. Transfer learning is a technique where a pre-trained model is used as the starting point for a new model. In this case, the pre-trained ResNet50 model is used as the base for a new model that is trained to classify the chest x-ray images into three categories: normal, lung opacity, and pneumonia.

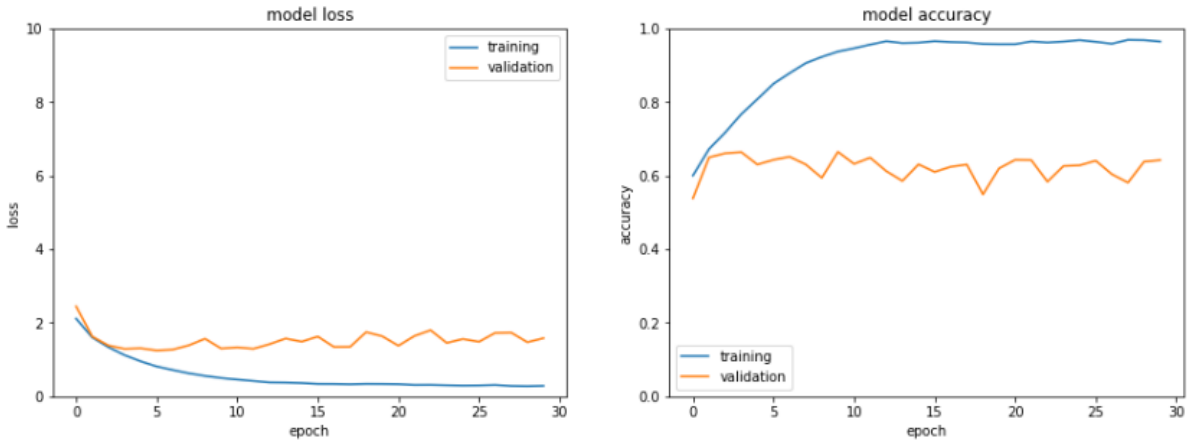
The first step is to preprocess the input data using the ResNet50 preprocess_input function. This function applies the necessary preprocessing to the input data so that it can be used with the ResNet50 model.

Next, a new model is defined using the Sequential API in Keras. The ResNet50 model is added to the new model as the first layer, followed by a Flatten layer to convert the output of the ResNet50 model to a one-dimensional array. Then, two fully connected layers are added, each followed by a batch normalization layer and a dropout layer to prevent overfitting. The final layer is a dense layer with a softmax activation function to produce the final classification output.

The model is compiled using the Adam optimizer with a learning rate of 0.0001 and categorical cross-entropy as the loss function. The model is then trained on the preprocessed training data for 30 epochs with a batch size of 256, and the validation data is used for validation during the training.

Finally, the training and validation loss and accuracy are plotted using Matplotlib, and the model is saved using the pickle module.

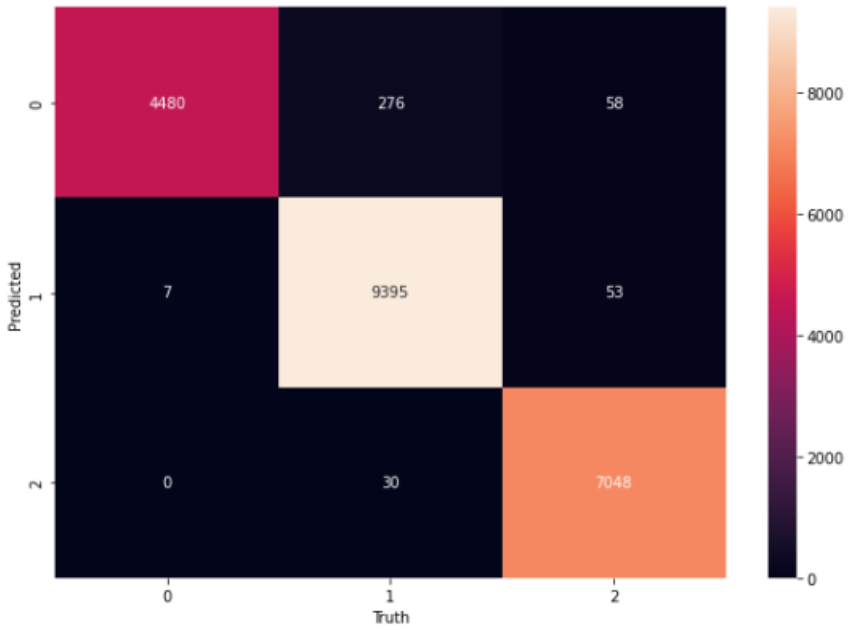
The resulting model achieves an accuracy of 97% on the training data and 64% on the test data.



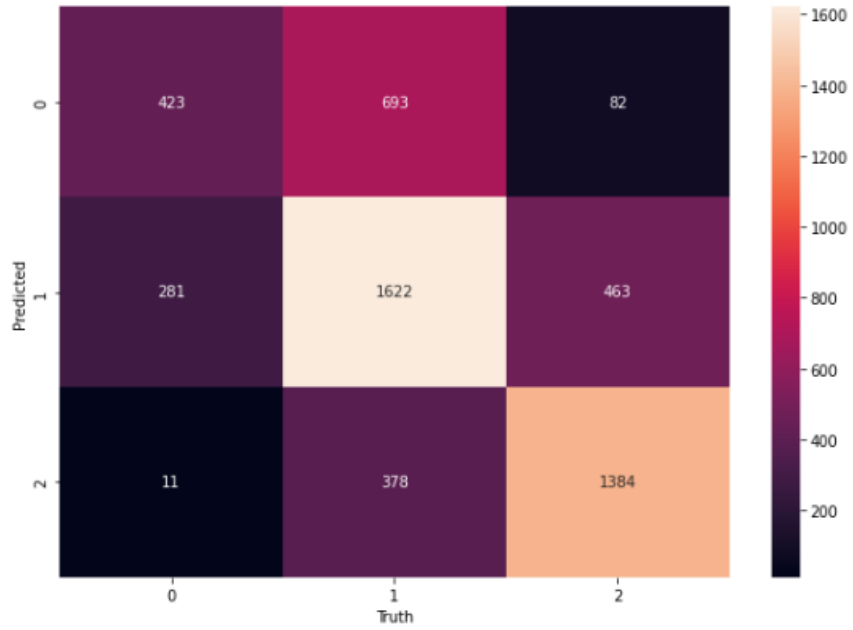
Then code uses a trained ResNet50 model on the pneumonia dataset to make predictions on both the training and testing data, calculates accuracy metrics, generates confusion matrices and a classification report, and stores the results in an Excel file. The data is preprocessed using **`preprocess_input`**, and the trained model is loaded using **`pickle.load`**. The predicted output is

compared with the actual output using `classification_report` and `confusion_matrix`, and the accuracy is calculated using `diagonal` and `sum`. The results are then stored in a dictionary and appended to an Excel file named `df_results.xlsx`.

```
668/668 [==============================] - 25s 23ms/step
              precision    recall  f1-score   support

           0       0.93      1.00      0.96      4487
           1       0.99      0.97      0.98      9701
           2       1.00      0.98      0.99      7159

    accuracy                           0.98     21347
   macro avg       0.97      0.98      0.98     21347
weighted avg       0.98      0.98      0.98     21347
```



```
167/167 [==============================] - 4s 24ms/step
              precision    recall  f1-score   support

           0       0.35      0.59      0.44       715
           1       0.69      0.60      0.64      2693
           2       0.78      0.72      0.75      1929

    accuracy                           0.64      5337
   macro avg       0.61      0.64      0.61      5337
weighted avg       0.68      0.64      0.65      5337
```



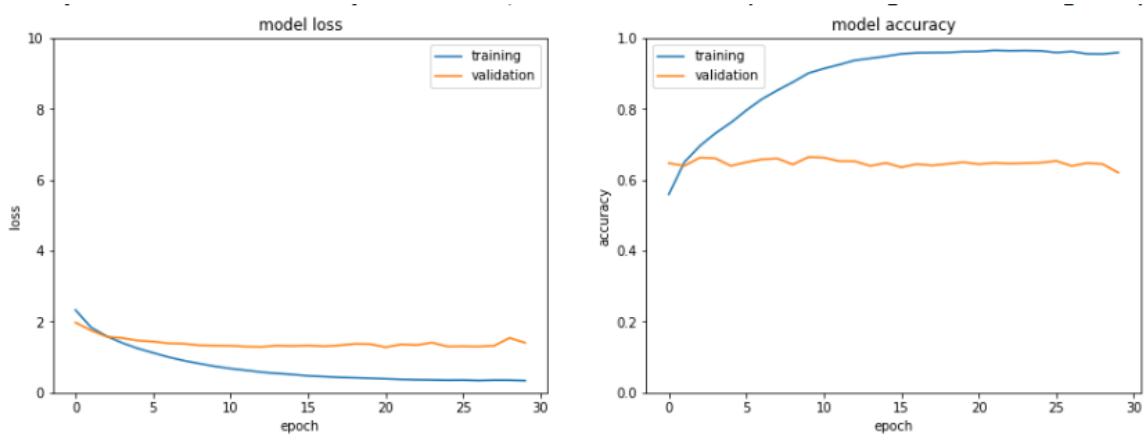
VGGNet16

Transfer learning using VGG16 model is performed on the pneumonia detection dataset. The images are preprocessed using the preprocess_input function from Keras. Then, a function CNN_with_Vggnet is defined which creates a sequential model with a pre-trained VGG16 model as its base. The last layer of the base model is removed, and a flatten layer, a dense layer with 64 units, batch normalization, dropout, and a final dense layer with 3 units are added. The model is compiled with Adam optimizer with a learning rate of 0.0001, and categorical cross-entropy loss.

The model is trained for 30 epochs with a batch size of 256 on the preprocessed training images and their corresponding labels. The validation data is passed as a tuple of preprocessed testing images and their labels. The model history is visualized using two subplots; the first plot shows the model loss for both training and validation data, and the

second plot shows the model accuracy for both training and validation data. The model_TL_Vgg16.pkl file is pickled using pickle.dump for later use.

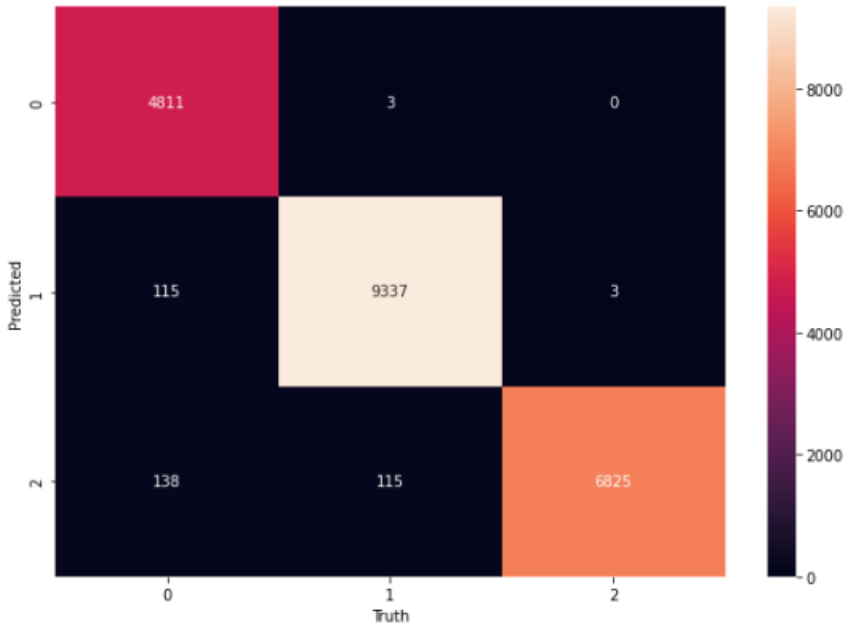
Finally, the model is evaluated on the test data, and the classification report, confusion matrix, and accuracy scores are computed and stored in a dataframe called df_results. The overall test accuracy is 96%, and the test accuracy for 'Lung Opacity' and 'Normal' classes are 62%.



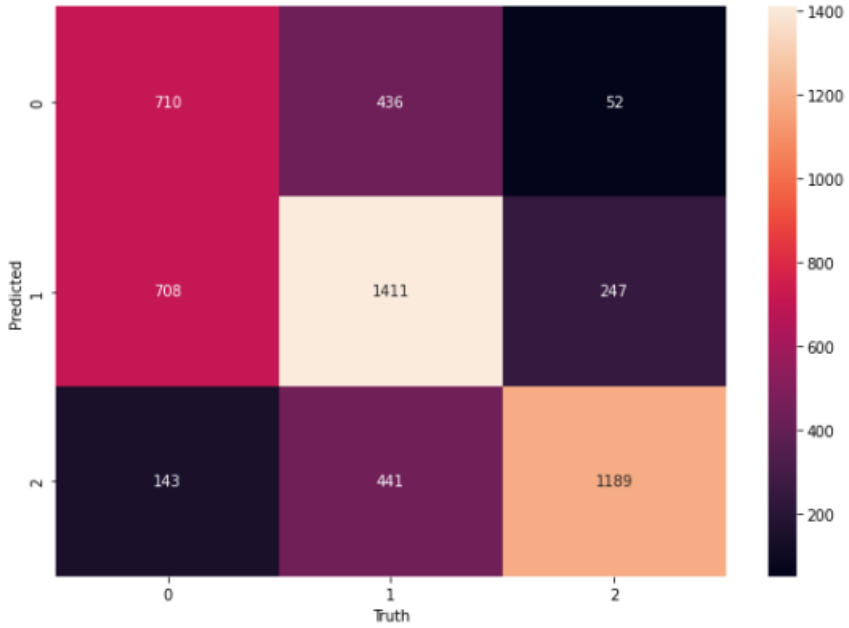
Then we are evaluating the performance of a pre-trained VGG16 model on the task of classifying chest x-ray images into three categories: normal, pneumonia, or lung opacity.

The results of the evaluation are stored in a dictionary and added to an Excel file containing the results of previous evaluations. The dictionary includes information such as the model name, overall training and test accuracy, and accuracy for each class.

```
...vars
668/668 [==============================] - 23s 22ms/step
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      5064
           1       0.99      0.99      0.99      9455
           2       0.96      1.00      0.98      6828

    accuracy                           0.98     21347
   macro avg       0.98      0.98      0.98     21347
weighted avg       0.98      0.98      0.98     21347
```

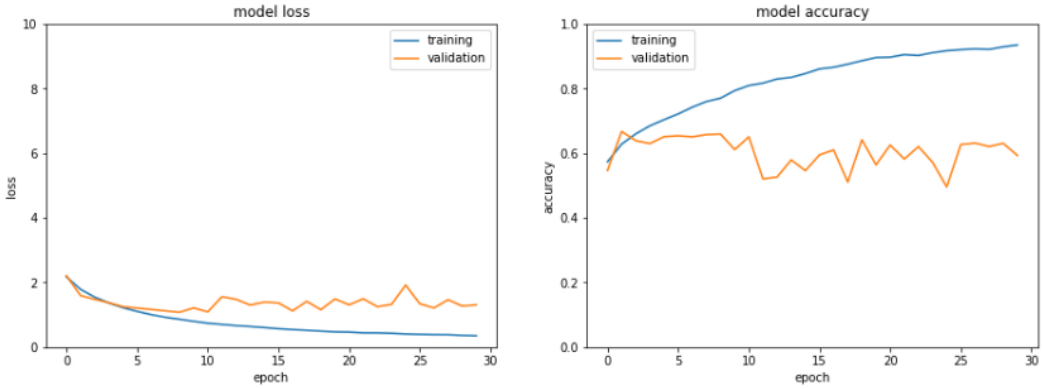


```
167/167 [==============================] - 4s 25ms/step
              precision    recall  f1-score   support

           0       0.59      0.45      0.51      1561
           1       0.60      0.62      0.61      2288
           2       0.67      0.80      0.73      1488

    accuracy                           0.62      5337
   macro avg       0.62      0.62      0.62      5337
weighted avg       0.62      0.62      0.61      5337
```



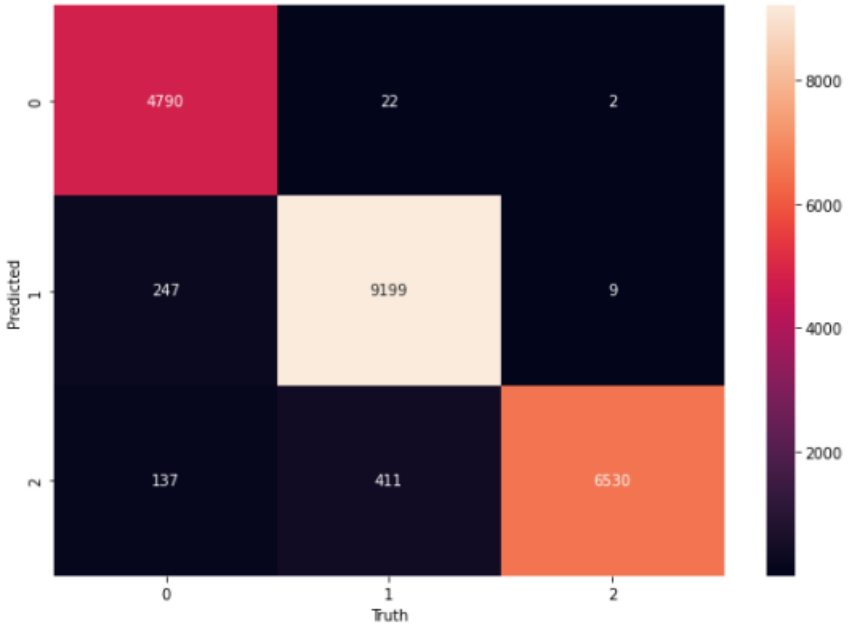
InceptionV2

Here we are working with a convolutional neural network (CNN) with transfer learning using the InceptionResNetV2 architecture. The code loads and preprocesses the training and testing data, defines the CNN architecture, and trains the model using the training data. It then plots the loss and accuracy for both the training and validation data. Finally, it saves the trained model as a pickle file.

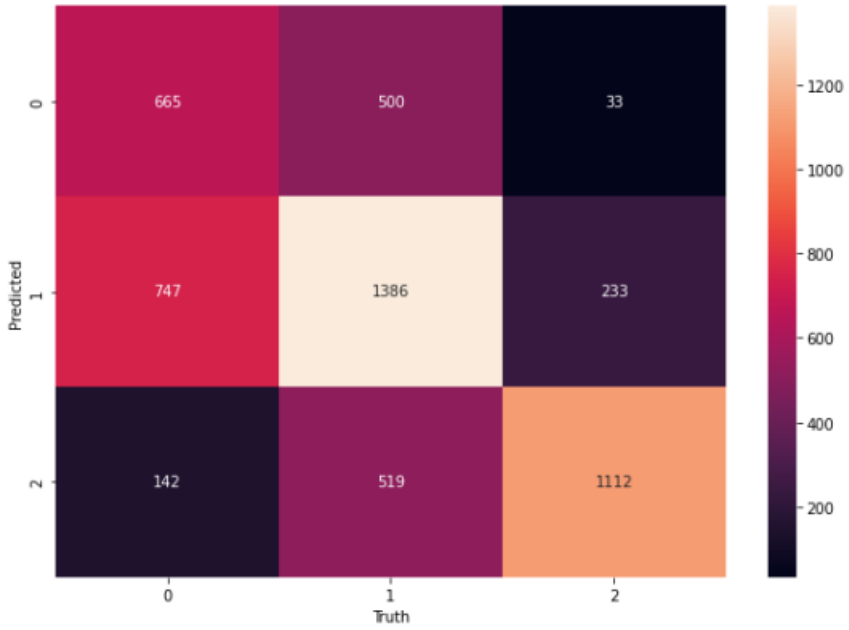


Then we load a pickled transfer learning InceptionResNetV2 model and uses it to make predictions on both the training and test data. It calculates and displays classification reports and confusion matrices for both the training and test data. It also calculates and stores accuracy numbers for both the training and test data in a pandas dataframe, along with the name of the model used. The dataframe is then saved to an Excel file.

```
668/668 [==============================] - 43s 47ms/step
              precision    recall  f1-score   support

           0       1.00      0.93      0.96      5174
           1       0.97      0.96      0.96      9632
           2       0.92      1.00      0.96      6541

    accuracy                           0.96     21347
   macro avg       0.96      0.96      0.96     21347
weighted avg       0.96      0.96      0.96     21347
```



```
167/167 [==============================] - 8s 50ms/step
              precision    recall  f1-score   support

           0       0.56      0.43      0.48      1554
           1       0.59      0.58      0.58      2405
           2       0.63      0.81      0.71      1378

    accuracy                           0.59      5337
   macro avg       0.59      0.60      0.59      5337
weighted avg       0.59      0.59      0.58      5337
```



Classification Model Results

Now , we compare the performance of each model. we have also sorted the models based on their test accuracy and displayed the results.

Resnet50 based Classification model performed the best with 64% test accuracy and 98% train accuracy.

Model	Tr_Acc Overall	Ts_Acc Overall
TL Resnet50	0.980138	0.642496
TL Vgg16	0.982480	0.620199
TL InceptionV2	0.961212	0.592655
CNN Tuned	0.977514	0.581413
CNN Lenet	1.000000	0.578790

Model	Tr_Acc Overall	Tr_Acc Lung Opacity	Tr_Acc No Lung Opacity / Not Normal	Tr_Acc Normal	Ts_Acc Overall	Ts_Acc Lung Opacity	Ts_Acc No Lung Opacity / Not Normal	Ts_Acc Normal
TL Resnet50	0.980138	0.930619	0.993654	0.995762	0.642496	0.353088	0.685545	0.780598
TL Vgg16	0.982480	0.999377	0.987520	0.964255	0.620199	0.592654	0.596365	0.670615
TL InceptionV2	0.961212	0.995015	0.972924	0.922577	0.592655	0.555092	0.585799	0.627186
CNN Tuned	0.977514	0.941836	0.998414	0.973863	0.581413	0.281302	0.726965	0.589961
CNN Lenet	1.000000	1.000000	1.000000	1.000000	0.578790	0.407346	0.585376	0.685843

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

RCNN - Selective Search

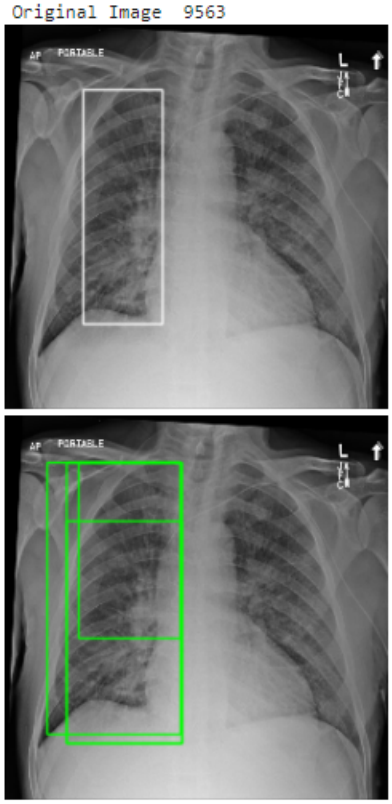
The selective search approach using OpenCV's createSelectiveSearchSegmentation() function is a popular method for generating region proposals. The approach involves segmenting an image into different regions based on color, texture, and other features, and then grouping these regions to form potential object regions. It can be computationally expensive as it generates a large number of potential regions, but it helps to reduce the number of regions processed by a classifier.

In the code, we have implemented the calculate_iou() function to compute the Intersection over Union (IoU) metric between the ground truth bounding box and the bounding box generated by the selective search algorithm. We also set a threshold of 0.5 for the IoU to decide whether to consider a bounding box as a positive example or not.

We have processed an input image using selective search and have visualized the region proposals generated by the algorithm. The code successfully identified bounding boxes with a high IoU value for a specific image.

It's worth noting that selective search can generate many potential regions, and not all of them will be relevant to the object of interest. Therefore, combining selective search with a classification model, such as Faster R-CNN, can further refine the regions and improve the accuracy of the detection.

Original Image 9563

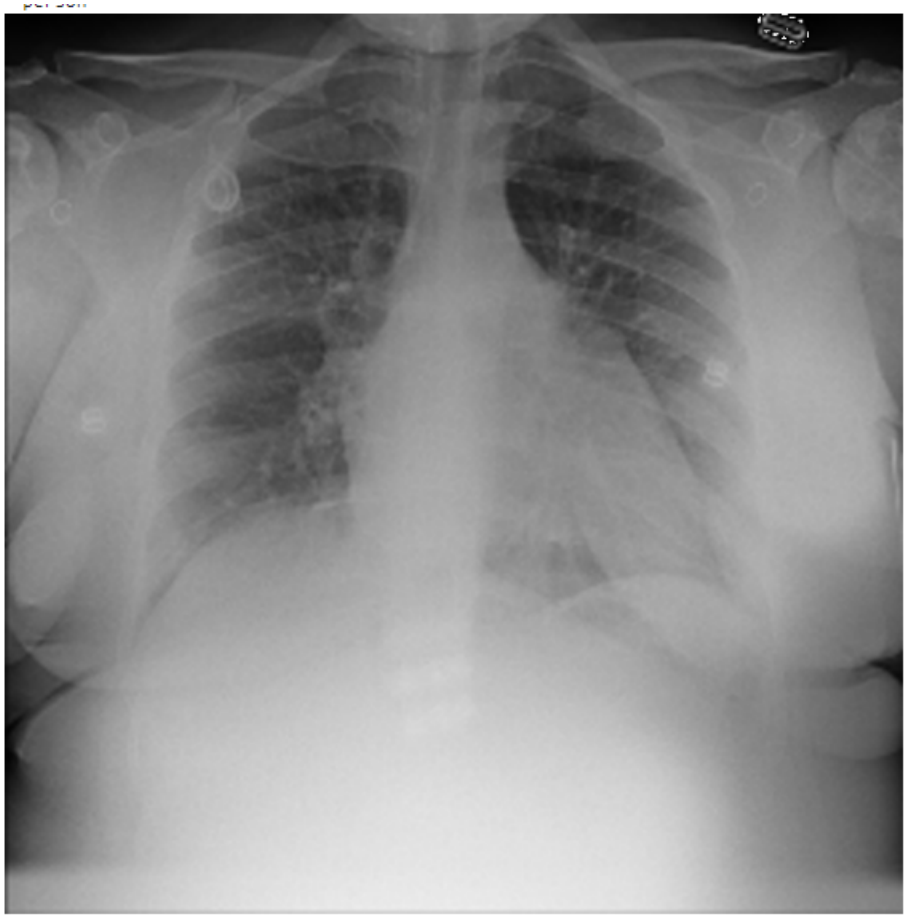


No. of BBs: 4 , iou 0.5742065421209807 , regions 313

Faster RCNN - Using gluoncv

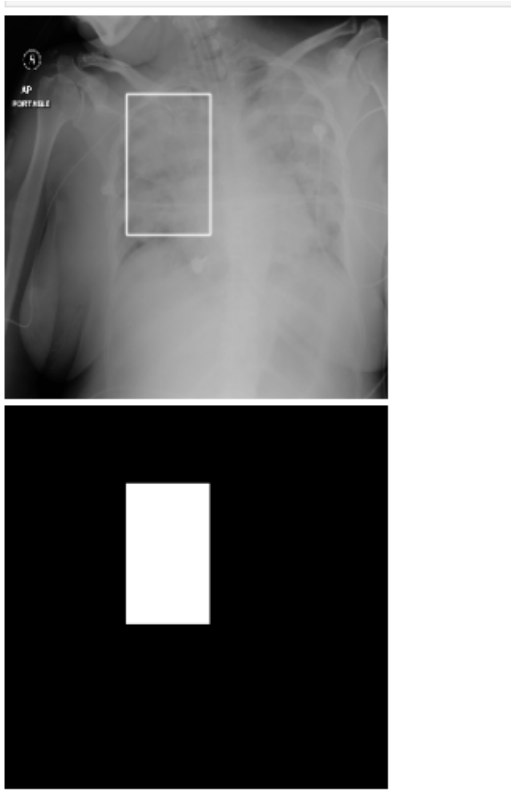
We are using the GluonCV library to implement a pre-trained Faster R-CNN model with ResNet50 as the base model. The code reads in an image and then applies the model to obtain bounding box predictions, class IDs, and confidence scores for the objects in the image.

we have identified that the model is detecting 'person' as one of the classes.



UNET - Use binary segmentation for detection

In this step, a UNET architecture for image segmentation is being implemented for identifying pneumonia in chest X-rays by training the network on a dataset consisting of chest X-ray images and their corresponding binary segmentation masks, and a helper function is defined to extract the bounding box coordinates for each image in the dataset to use as input for the UNET model; the code also includes a sample image plot to visualize the overlay of the bounding box on the X-ray image and the corresponding segmentation mask.



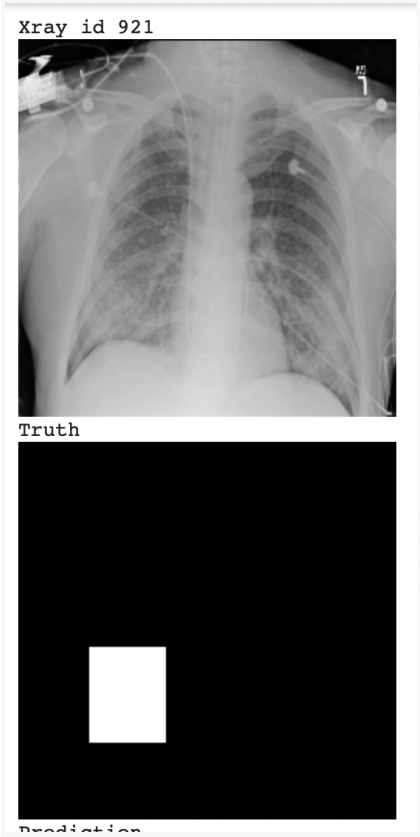
Further we are implementing a UNET model for image segmentation using TensorFlow and Keras libraries. The input image is first preprocessed by converting it to a 3-channel RGB image, and then the MobileNetV2 architecture is used as the encoder to extract features from the input image. The MobileNetV2 model is initialized with pre-trained weights from the ImageNet dataset and the include_top parameter is set to False, meaning that the last few layers of the model that are responsible for classification are not included.

In the UNET model, the encoder output is connected to the decoder part of the network through a series of skip connections. The names of the layers that are used for the skip connections are specified in the skip_connection_names list. These layers are then accessed from the encoder model and concatenated with the upsampled output from the previous decoder layer. The decoder part of the network consists of a series of Conv2D layers with batch normalization and ReLU activation functions. The output layer of the model is a single-channel image that represents the segmented image mask.

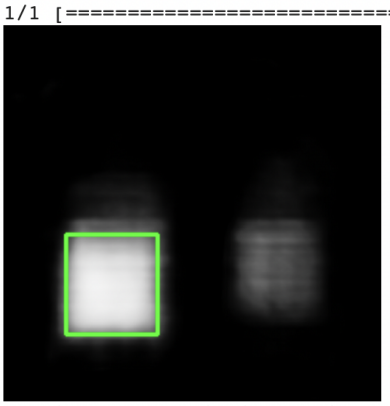
To train the model, the binary cross-entropy loss function and the dice coefficient metric are used. The model is compiled with the Adam optimizer, and the loss and metric are set as the binary cross-entropy and dice coefficient functions, respectively. Additionally, callbacks such as ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau are used to save the best model weights, stop the training early if the model is not improving, and reduce the learning rate if the model is not converging, respectively.

Finally, the model is trained on the input RGB images (X_rgb) and the corresponding binary masks (Y_mask) using a batch size of 32 and for 20 epochs with 10% of the data used for validation.

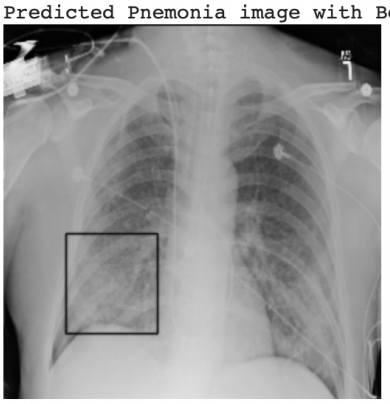
In the last cell of this section, we select a random image from a dataset of pneumonia cases and display the image along with the ground truth mask and the predicted mask generated by a UNet model. The contour is detected using OpenCV api and the bounding box is finally overlayed on the predicted mask image.



```
Prediction
1/1 [==============================] - 0s 28ms/step
```



```
Predicted Pnemonia image with Bounding Box
```



Design a clickable UI based interface which can browse & input the image, output the class & bounding box or mask (highlight area of interest) of the input image

This was the optional part. Though the code was implemented, but there is a limitation on the Tkinter side to launch browzer windows on Google collab.

Team did move to Jupyter notebook just for this task, but due to lack of time the implementation could not be completed (file path's and model pickle file path/locations need to be moved).

4. Model Evaluation

Our final model is the one based out of U-NET architecture for image segmentation. The model consists of one encoder(for upsampling) and followed by a decoder(for downsampling). We have imported the U-net model "MobileNetV2" as backbone and loaded weights of image net. The advantages of using MobileNetV2 is that it has less parameters (easy to train), the pre-trained encoder helps the model to converge much faster and to achieve high performance.

The dataset is split into training, validation and testing using 80:10:10 ratio. The encoding layer will be created by MobileNetV2 and the decoder will consist of upsampling layers which helps to get the desired mask shape. The decoder also needs feature maps from specific layers of the encoder. The upsampling is done by using the Conv2D layer. The batch normalization and non-linearity are added to each step of the upsampling. Each stage of the decoder is concatenated with the corresponding layer of the encoder.

The dice coefficient is used as the metric to measure the performance and the dice coefficient loss. The Adam optimizer with dice coefficient loss is used to train the model. The callbacks ReduceLROnPlateau and EarlyStoppong are used during the model training.

```
def unet_MN_based_model():
  inputs = Input(shape=(256, 256, 3), name='input_image')
  encoder = MobileNetV2(input_tensor=inputs, weights='imagenet', include_top=False,
alpha=0.35)
  skip_connection_names =
['input_image','block_1_expand_relu','block_3_expand_relu','block_6_expand_relu']
  encoder_output = encoder.get_layer('block_13_expand_relu').output
  f = [16, 32, 48, 64]
  x = encoder_output
  for i in range(1,len(skip_connection_names)+1, 1):
        x_skip = encoder.get_layer(skip_connection_names[-i]).output
        x = UpSampling2D((2, 2))(x)
        x = Concatenate()([x, x_skip])
        x = Conv2D(f[-i], (3, 3), padding="same")(x)
```

```
            x = BatchNormalization()(x)
            x = Activation("relu")(x)
            x = Conv2D(f[-i], (3, 3), padding="same")(x)
            x = BatchNormalization()(x)
            x = Activation("relu")(x)
    x = Conv2D(1, (1, 1), padding="same")(x)
    x = Activation("sigmoid")(x)
    model = Model(inputs, x)
    return model

model_unet = unet_MN_based_model()
model_unet.summary()

def dice_coefficient(y_true, y_pred):
            numerator = 2 * tf.reduce_sum(y_true * y_pred)
            denominator = tf.reduce_sum(y_true + y_pred)

            return numerator / (denominator + tf.keras.backend.epsilon())


model_unet.compile(optimizer='Adam', loss=binary_crossentropy, metrics=[dice_coefficient])

checkpoint = ModelCheckpoint("model-{loss:.2f}.h5", monitor="loss", verbose=1,
save_best_only=True,
                             save_weights_only=True, mode="min")
stop = EarlyStopping(monitor="loss", patience=5, mode="min")
reduce_lr = ReduceLROnPlateau(monitor="loss", factor=0.2, patience=5, min_lr=1e-6,
verbose=1, mode="min")

#fit the model
model_unet.fit(X_rgb, Y_mask, batch_size=32, epochs=25, validation_split=0.1)
```

The trained model on the test dataset got 59% dice coefficient

5. Comparison To Benchmark

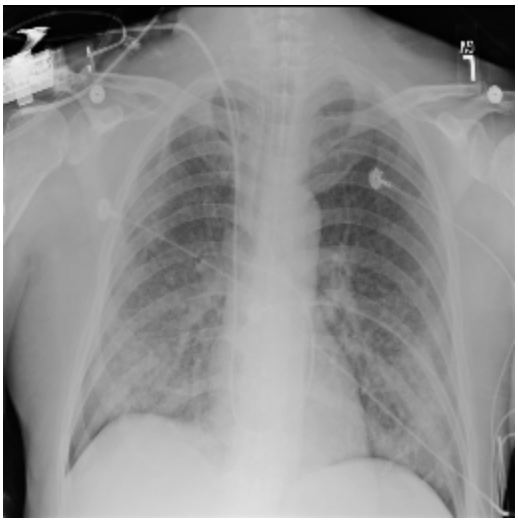
For the Classification task, we improved from 57-58% to 64% using various Transfer Learning techniques

For Detection, 3 models were evaluated. Out of them, U-Net was the most effective with ~60% accuracy (dice-coefficient score).

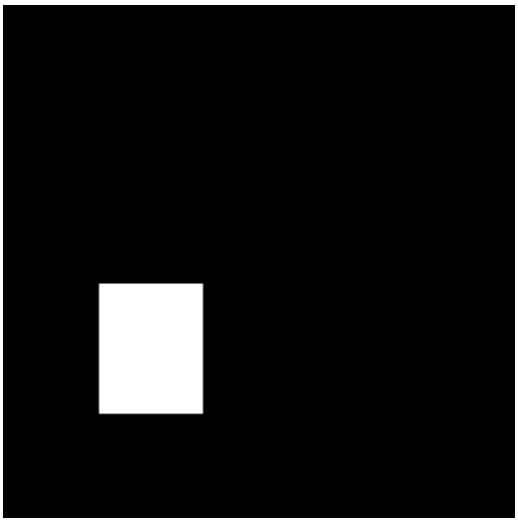
S.No	Model	Accuracy in %
1	CNN (Resnet50) - Classification	64
2	RCNN - Detection	57
3	Faster RCNN - Detection	49
4	U-Net - Detection	59

6. Visualizations

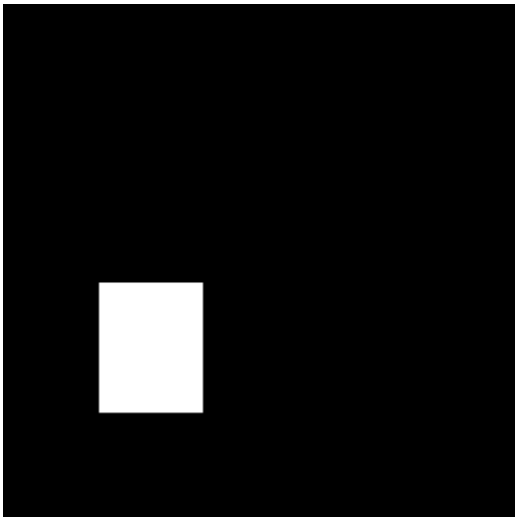
Input Image:



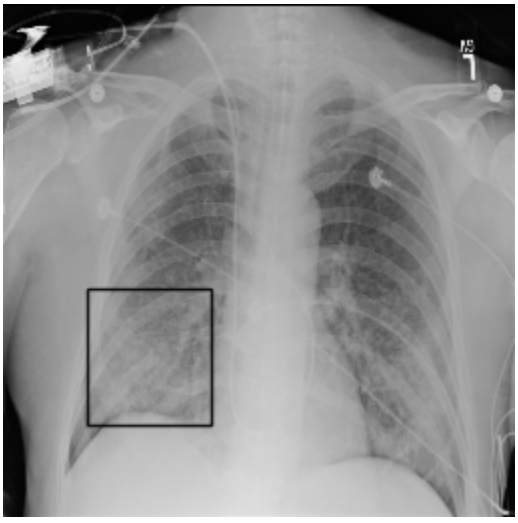
Truth:



Predicted:



Output:



7. Implications

Our solution can be used to develop a pneumonia detection system that can help clinicians make accurate and timely diagnosis. The system can analyze medical images such as chest X-rays or CT scans and detect the presence of pneumonia based on the patterns and features in the images. The system can also be trained on large datasets of medical records to identify the risk factors and symptoms associated with pneumonia, which can help clinicians make better-informed decisions.

One of the key benefits of our solution using CNN Model, Transfer Learning Model (RESNET50, VGGNet16, InceptionV2), RCNN, UNET for pneumonia detection is that it can improve the accuracy and speed of diagnoses, which can lead to better patient outcomes. Additionally, it can help identify cases of pneumonia that may be missed by human clinicians, particularly in cases where the condition is not yet advanced or the clinician is inexperienced.

To develop a reliable deep learning based pneumonia detection system, it is important to use high-quality medical images and large datasets of medical records. The system should also be validated using independent datasets to ensure its accuracy and generalizability. In terms of confidence level, the system should be validated to achieve a high sensitivity and specificity to minimize false positives and false negatives.

In addition to developing an ML based pneumonia detection system, it is important to have appropriate protocols and guidelines in place to ensure the system is used effectively and safely in clinical settings. Clinicians should also be trained to use the system correctly and interpret its results appropriately. However, careful consideration should be given to the design, training, and validation of deep learning models to ensure their safety and effectiveness in clinical settings.

8. Limitations

- Large volumes of data are necessary for deep learning. Additionally, the more accurate and powerful models will need more parameters, which calls for more data.
- Deep learning models are rigid and incapable of multitasking after they have been trained. Only one unique problem can they effectively and precisely solve. Even resolving a comparable issue would need system retraining.
- Even with vast amounts of data, existing deep learning approaches cannot handle any application that needs thinking, like programming or using the scientific method. They are also utterly incapable of long-term planning and algorithmic-like data manipulation.
- Biases are another significant concern with deep learning algorithms. When a model is trained on biased data, it will replicate similar biases in its predictions. Deep learning programmers have struggled with this issue since models learn to distinguish based on minute differences in data pieces.

Where does model fall short in the real world

CNN Model in pneumonia detection is their potential for false positives and false negatives. CNN models can sometimes mistake other lung conditions or artifacts in the images for pneumonia or fail to detect pneumonia in some cases. This can lead to incorrect diagnoses or missed diagnoses, which can have serious consequences for patients. It can be computationally intensive and require high-performance computing resources, which may not be readily available in some clinical settings. This can limit the practicality and scalability of CNN models for pneumonia detection in real-world settings.

In the RCNN Model, RCNNs are their computational complexity. The multiple stages of the RCNN pipeline, including the region proposal network, convolutional layers, and pooling layers, can be computationally intensive and require high-performance hardware. This can limit the practicality and scalability of RCNN models in clinical settings where resources are limited.

Transfer Learning models can be computationally intensive, especially when fine-tuning is used to adapt the pre-trained model to the pneumonia detection task. Fine-tuning requires re-training the last layers of the pre-trained model on the pneumonia detection dataset, which can be computationally expensive and may require high-performance hardware.

Finally, transfer learning models can be difficult to interpret, which can limit their usefulness in clinical decision-making. It may not always be clear why certain features were selected or how the model arrived at its decision.

Deep Learning models can be prone to overfitting, especially when the dataset size is small or unbalanced. Overfitting occurs when a model learns to memorize the training data rather than learning the underlying patterns and features. This can lead to poor generalization performance on new data and reduced accuracy in the real world.

In conclusion, deep learning models have shown great potential in pneumonia detection, but they also have limitations such as the need for large amounts of labeled data, potential for overfitting, computational complexity, difficulty in interpretation, and lack of explainability. To overcome these limitations, it is essential to carefully design and validate deep learning models, develop robust and reliable clinical workflows for their deployment in real-world clinical settings, and explore new methods for model explainability and interpretability.

What can you do to enhance the solution:

There are several approaches that can be used to enhance the solution for pneumonia detection using deep learning models:

- Collect and curate a large and diverse dataset: To improve the accuracy and generalizability of the deep learning model, it is important to collect a large and diverse dataset of medical images. The dataset should include images from different patient populations, imaging modalities, and clinical settings.
- Use data augmentation: Data augmentation techniques can be used to artificially increase the size of the dataset and introduce variability and diversity in the data. This can help to prevent overfitting and improve the generalizability of the deep learning model.
- Incorporate clinical data: Incorporating clinical data such as patient demographics, medical history, and laboratory test results can help to improve the accuracy and interpretability of the deep learning model. This can enable clinicians to better understand how the model arrived at its decision and make informed decisions about patient care.
- Develop explainable deep learning models: Explainable deep learning models can help to improve the interpretability and transparency of the model's decision-making process. This can enable clinicians to better understand how the model arrived at its decision and increase their trust in the model.

- Conduct thorough validation and testing: It is essential to conduct thorough validation and testing of the deep learning model on independent datasets to ensure that it performs well in real-world clinical settings. This can help to identify any limitations or potential biases in the model and enable further refinement and improvement.

In summary, enhancing the solution for pneumonia detection using deep learning models requires careful consideration of the dataset, model architecture, clinical data, and validation and testing procedures. By addressing these factors, it is possible to improve the accuracy, generalizability, interpretability, and trustworthiness of the deep learning model.

9. Closing Reflections

Learning from the process:

One of the key learnings from the process of pneumonia detection through deep learning models is the importance of dataset quality and size. Deep learning models require large and diverse datasets to be trained effectively, and the quality of the data is crucial for achieving high accuracy and generalizability of the model.

Another important learning is that transfer learning can be a powerful tool for pneumonia detection. Pre-trained models can be fine-tuned for pneumonia detection with relatively small amounts of data, reducing the need for extensive labeled data collection.

Furthermore, explainability and interpretability are important factors to consider when developing deep learning models for pneumonia detection. Medical professionals must be able to understand and interpret the outputs of these models, which can be challenging due to their complexity. Developing models that are explainable can help to build trust with clinicians and patients, ultimately leading to better decision-making.

Lastly, continuous evaluation and improvement are crucial for ensuring the accuracy and usefulness of deep learning models for pneumonia detection. Models should be regularly validated on new and diverse datasets to ensure that they perform well in a range of clinical settings.

In terms of what can be done differently next time:

One approach to improve pneumonia detection through deep learning models is to continue exploring new architectures and techniques for deep learning. For example, generative adversarial networks (GANs) and attention mechanisms have shown promise in the field of computer vision and could be applied to pneumonia detection.

Additionally, collaborate with healthcare professionals and domain experts to ensure that the model is aligned with clinical needs and workflows. Involving clinicians and patients in the design and testing of the deep learning model can help to identify potential limitations and biases and improve the model's accuracy, interpretability, and usability.

Overall, the key to developing effective deep learning models for pneumonia detection is to continue learning from past successes and failures, while also exploring new techniques and collaborating with domain experts.

Team - CV1A

Name	**Email**
Balakumar Kaliyaperumal	balakumar.kaliyaperumal@gmail.com
Poonam Thakur	poonampoojathakur17@gmail.com
Georlin Babu	georlin.babu@gmail.com
Deepesh Tripathi	deepesh.tr@gmail.com
Shruti Paradkar	paradkarshruti9@gmail.com
Jagadeesh	vnjagadeesh@gmail.com