greatlearning
Learning for Life

Capstone Project on
Pneumonia Detection

INTERIM REPORT

Participants

- Mrs Shilpa Kulkarni
- Mr Jignesh Mistry
- Mr Samit Biswas
- Mr Aarish Shahab

Pneumonia and Diagnosis

- Pneumonia is an infection of the lungs caused by bacteria, viruses and fungi. Pneumonia causes inflammation in the air sacs of one or both lungs which fill with fluid or pus, making it difficult to breathe and may result in fatality.

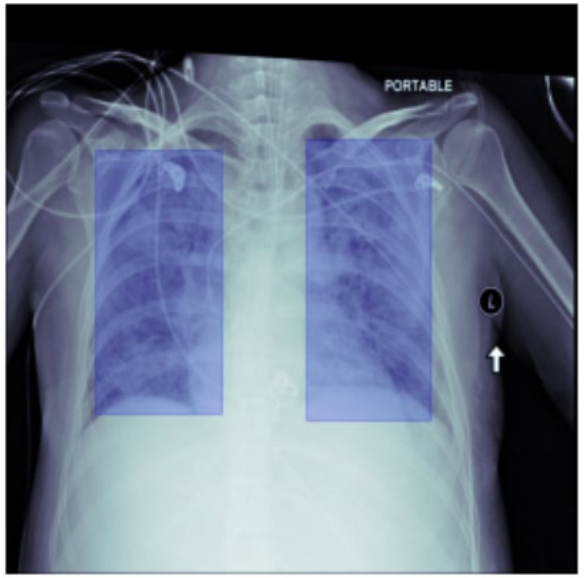
- World Health Organization (WHO) estimates, approximately 150 million lung infections and 4 million deaths can be attributed annually to Pneumonia. Small children and seniors are more susceptible to this infection. Fast, timely and accurate diagnosis is of utmost importance.

- Chest X-Ray (CXR) analysis is the first step in the diagnosis of Pneumonia.

Problem Statement:

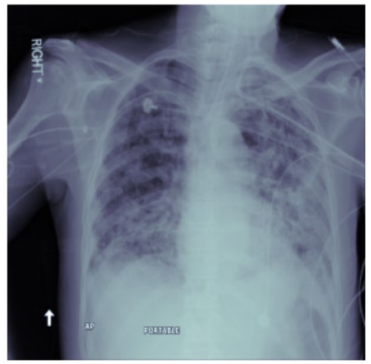
- The goal of this project is to build a model which detects if a given patient's CXR image indicates the presence of Pneumonia or not and also predicts the location of the Pneumonic patch. Both, accuracy and speed of detection are of utmost importance.

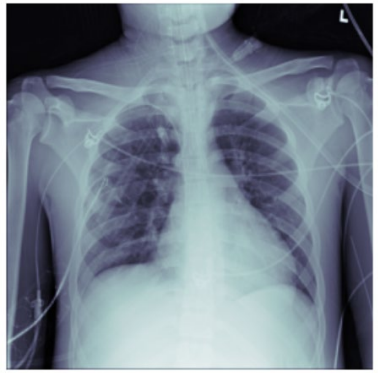
- To achieve these, we performed object (opacity) detection using a Neural Network(NN) with labelled samples of Chest X-Ray(CXR) images.



Data:

- Train and Test Images - The CXR images are present in the Digital Imaging and Communications in Medicine (DICOM) format, with .dcm file extension. They are separated into training and test sets





- Patients by class - A .csv file which has information about the class detected per patient is provided

	patientId	class
19529	b5234584-1487-492c-8742-444b9ca41c3d	Normal
807	0a8ccb49-debc-4e9a-b5dc-eefc3fe909ca	No Lung Opacity / Not Normal
1733	1851dff6-31ec-4453-9ddc-95c0aedace5b	Lung Opacity

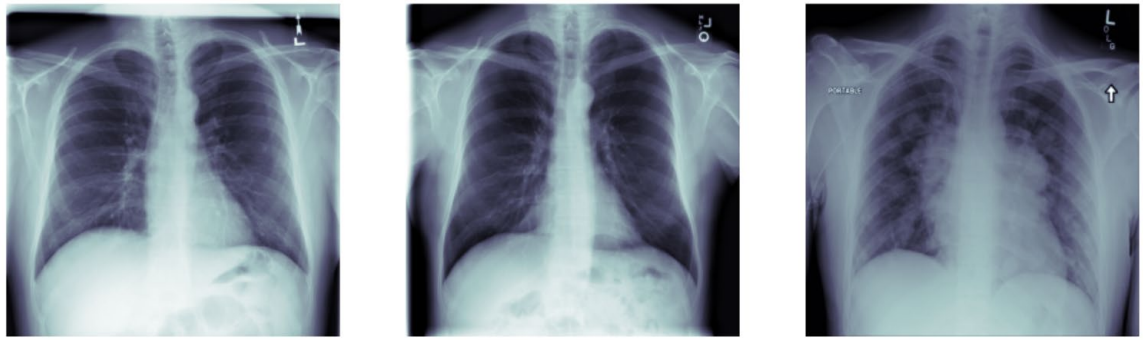
- Training labels – A .csv file with co-ordinates for each bounding box per patient is provided.

	patientId	x	y	width	height	Target
2508	3251dea8-4f74-4f4b-8f56-167b0213414b	140.0	583.0	188.0	114.0	1
2283	2feb2bad-6b36-4067-87f1-54d39235e3c5	247.0	503.0	163.0	155.0	1
24491	dc139a0e-0bbf-4ba4-8e71-c949022a542e	NaN	NaN	NaN	NaN	0

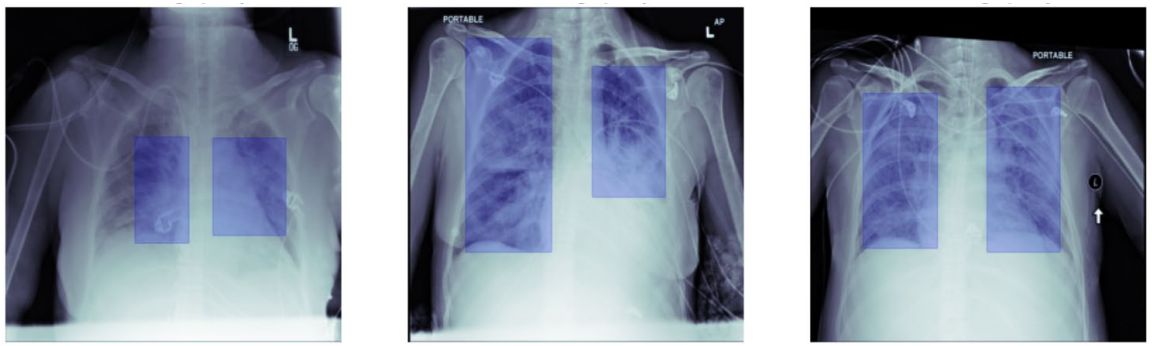
EDA, Pre-processing and findings

Approach to EDA:

- Row counts: There are same number of rows in both .csv files
  - There are multiple entries for some patients
- Target Values are 0 (Non-Pneumonic) and 1 (Pneumonic)
- 2 target values are mapped to 3 class values
  - Target 0 is mapped to “Normal” & “No Lung Opacity/Not Normal” classes

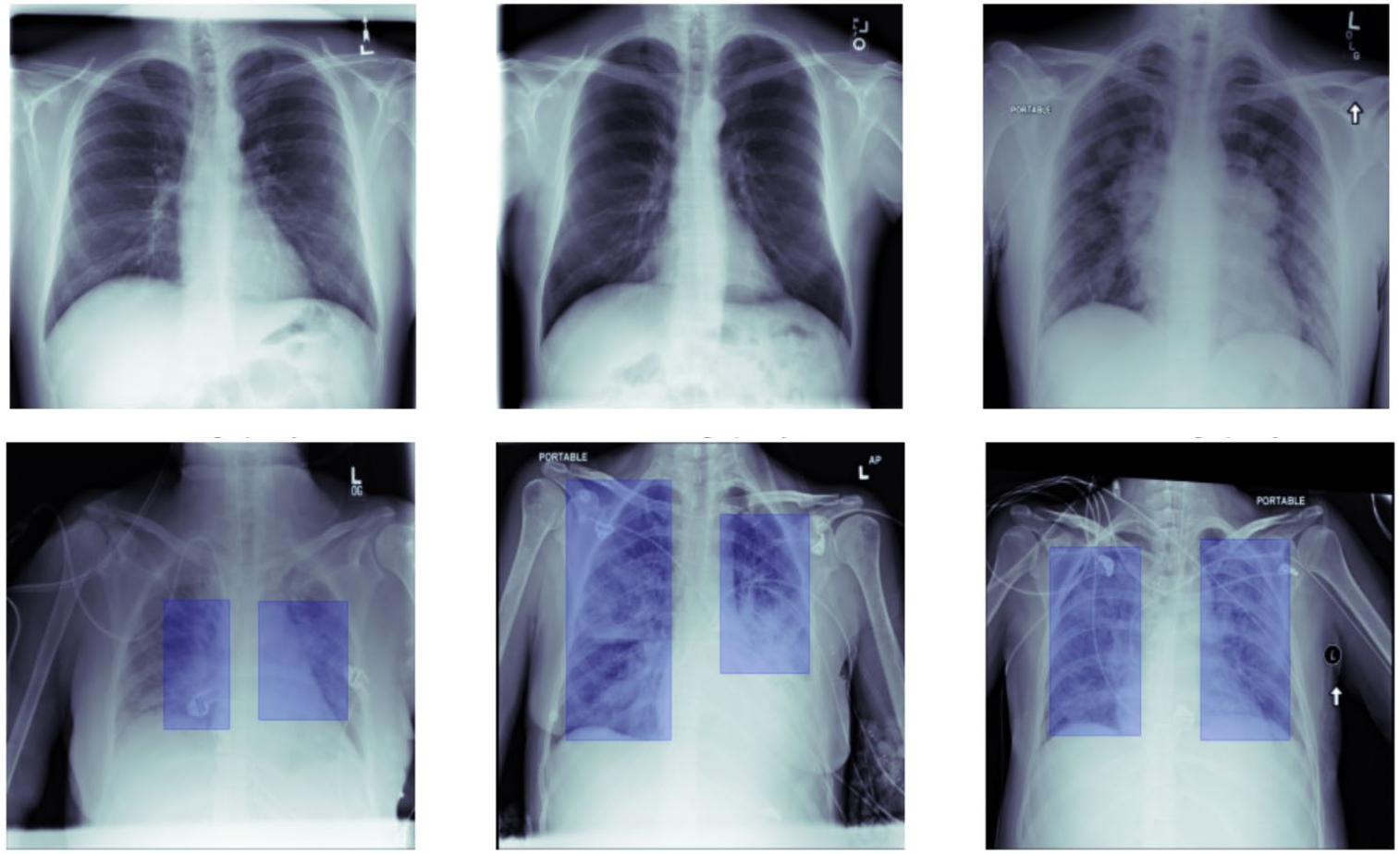


  - Target 1 is mapped to “Lung Opacity” class



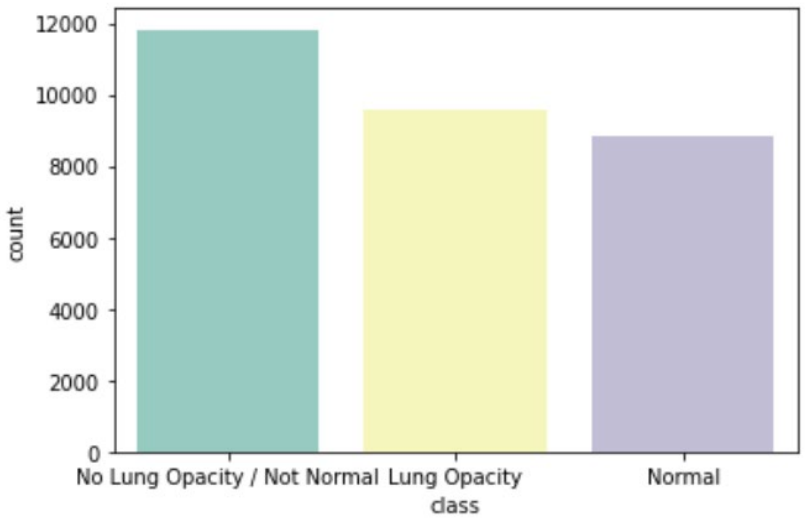
Approach to EDA:

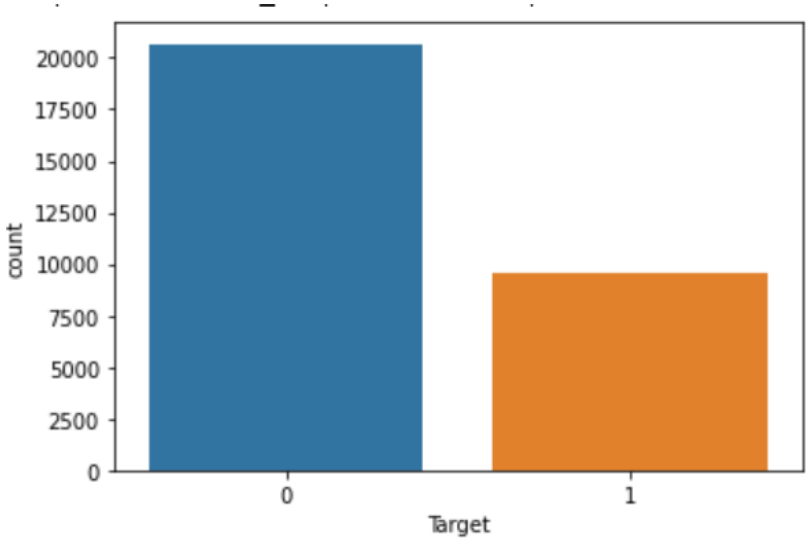
- Used pydicom package to load the .dcm images from the training dataset and visualized samples positive as well as negative cases.



Approach to EDA:

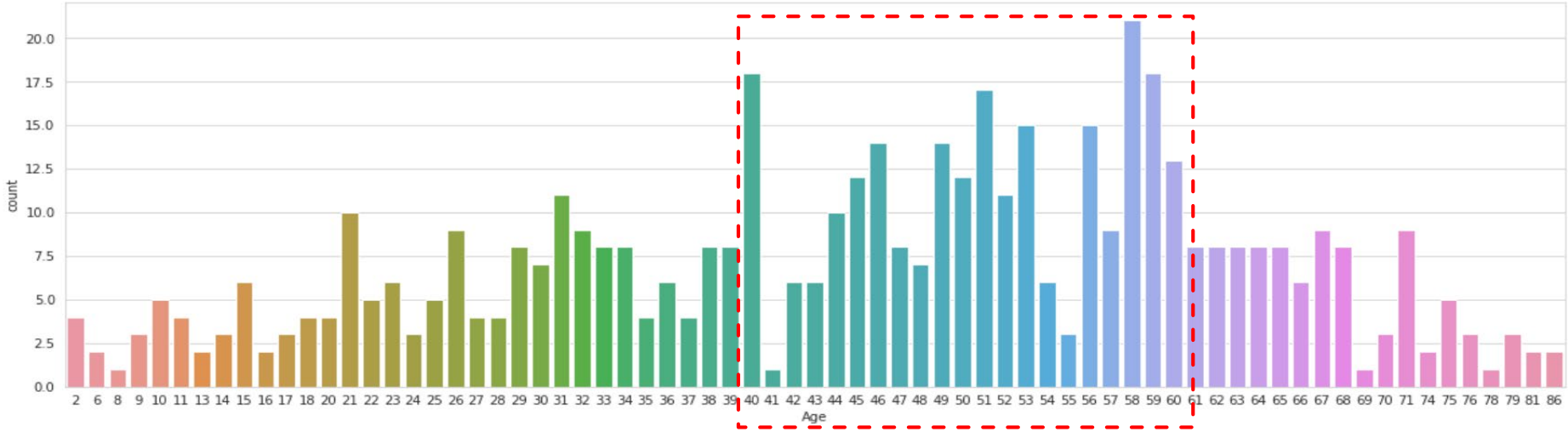
- Missing Values:
  - Class Details – There are no missing values
- Training Labels:
  - 68.3% of the data is missing values in columns x, y, width and height where Target = 0 (non-Pneumonia)
  - For 31.3% of data, there are no missing values where Target = 1
- Check for Class imbalance
  - No noticeable difference between row counts for classes (Normal, No Lung Opacity/Not Normal, Lung Opacity)
  - Target=0 includes 2 classes, only 31%.



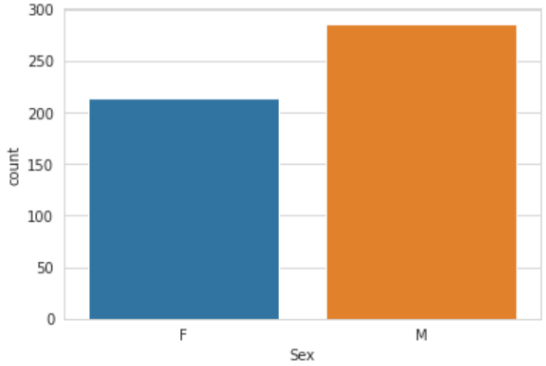


Approach to EDA:

- Maximum number of cases lies between 40 to 60 (for 500 random samples).
- Normally distributed.

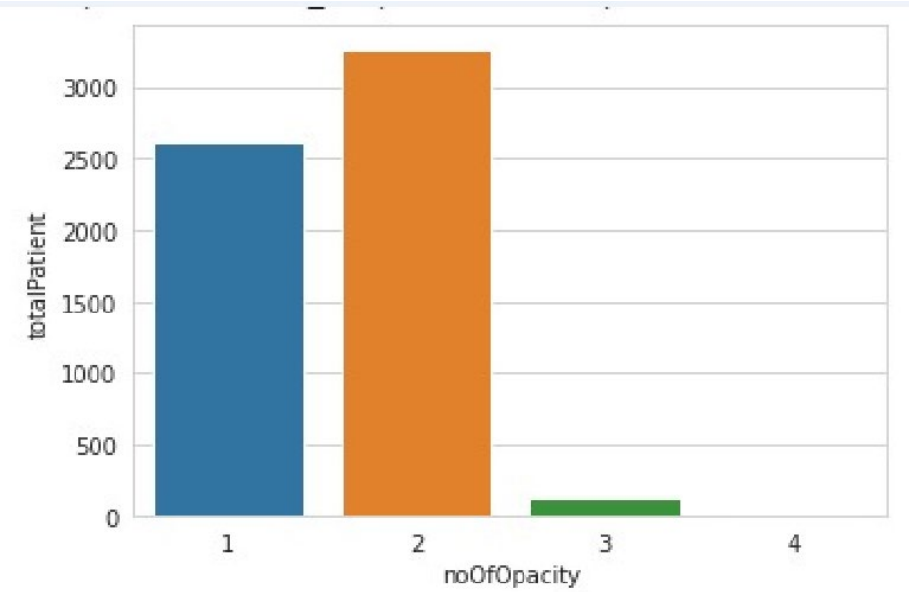


- Occurrence is more in male patients than female patients



Approach to EDA:

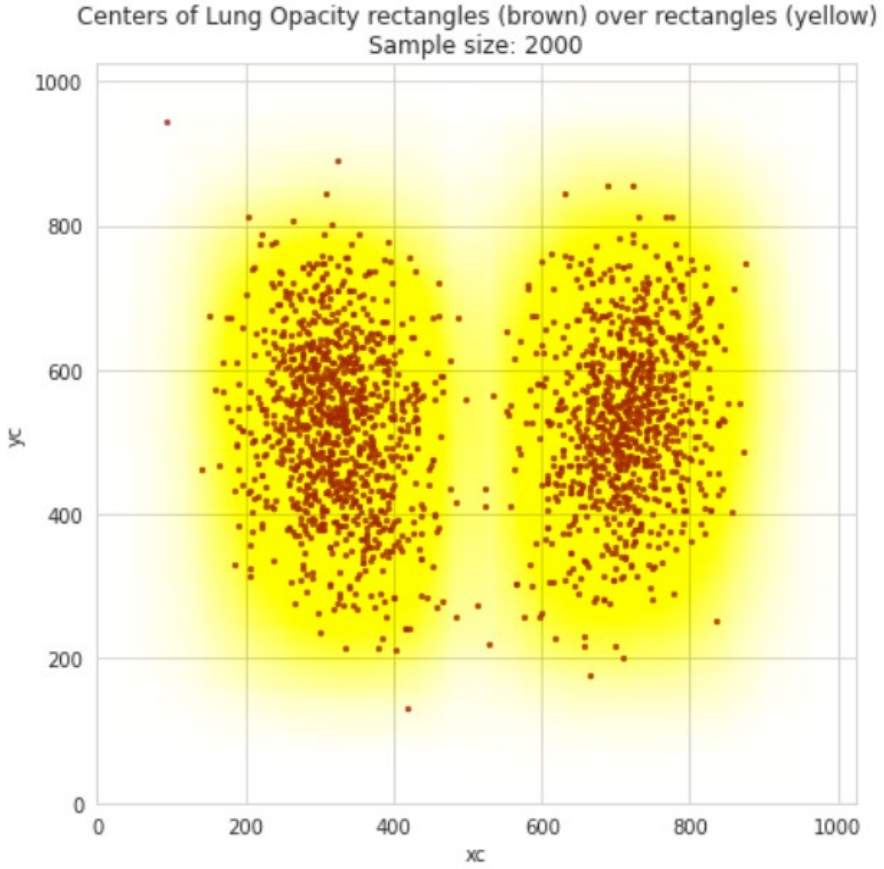
- Frequency of 2 bounding box (opacity) is higher

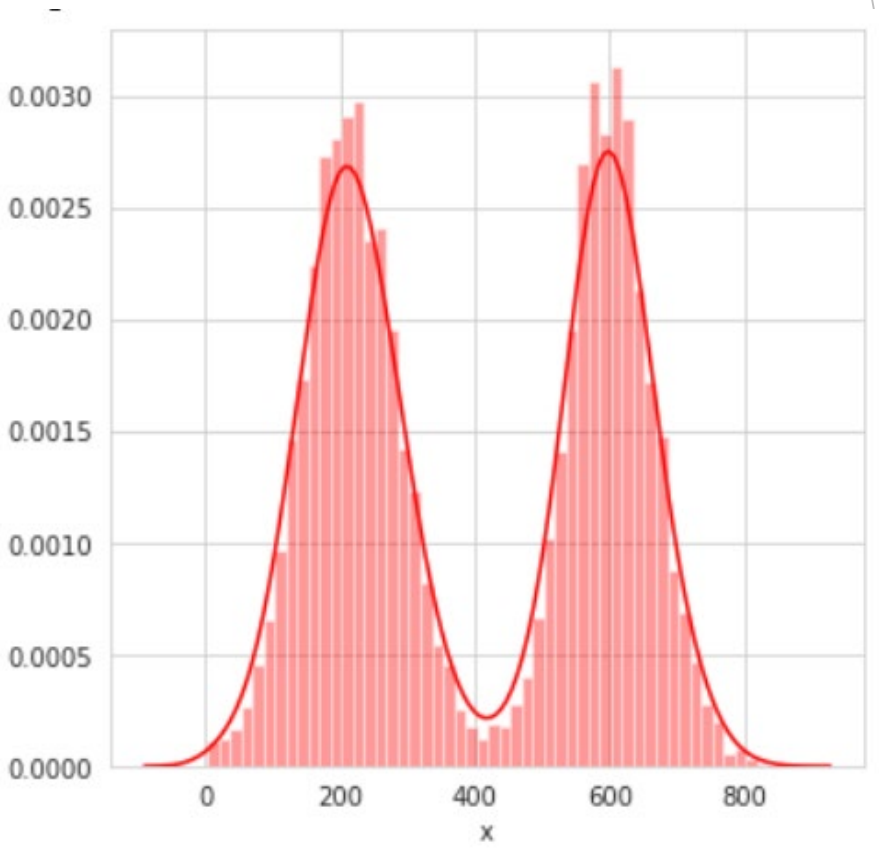


	noOfOpacity	Target	Class	totalPatient
0	1	0	No Lung Opacity / Not Normal	11821
1	1	0	Normal	8851
2	1	1	Lung Opacity	2614
3	2	1	Lung Opacity	3266
4	3	1	Lung Opacity	119
5	4	1	Lung Opacity	13

Approach to EDA:

- Analyse the Lung Opacity Window
  - For 2000 samples of training data, plotted the centres of the bounding boxes (bbox) for lung opacities over the actual boxes.
  - Probability is pneumonic patch is more around the centre of both the lungs .





Data Pre-processing

```
# Data Generator using Sequence for multiprocessing
from tensorflow.keras.utils import Sequence
class DataSequence(Sequence):
  def __init__(self, folder, filenames, pneumonia_locations=None, batch_size=BATCH_SIZE,
               image_size=IMAGE_SIZE, shuffle=True, augment=False, predict=False):
    self.folder = folder
    self.filenames = filenames
    self.pneumonia_locations = pneumonia_locations
    self.batch_size = batch_size
    self.image_size = image_size
    self.shuffle = shuffle
    self.augment = augment
    self.predict = predict
    self.on_epoch_end()

  def __load__(self, filename):
    # Load DICOM file as numpy array
    img = pydicom.dcmread(self.folder + filename).pixel_array

    # Create empty mask
    mask = np.zeros(img.shape)
    filename = filename.split('.')[0] # Remove the file extension
    if filename in pneumonia_locations:
      for location in pneumonia_locations[filename]:
        x, y, width, height = location
        mask[x:x+width, y:y+height] = 1 # Broadcast value of 1 to all the pixels within

    # If augment flag is on, then flip the image and mask horizontally half the times
    if self.augment and random.random() > 0.5:
      img = np.fliplr(img)
      mask = np.fliplr(mask)

    # Resize both the image and mask
    img = resize(img, (self.image_size, self.image_size), mode='symmetric')
    mask = resize(mask, (self.image_size, self.image_size), mode='symmetric') > 0.5

    # Add the channel dimension to both the image and masks files
    img = np.expand_dims(img, -1)
    mask = np.expand_dims(mask, -1)
```

Data Pre-processing

```
def __load_predict__(self, filename):
  # Load DICOM file as numpy array
  img = pydicom.dcmread(self.folder + filename).pixel_array
  img = resize(img, (self.image_size, self.image_size), mode='symmetric')
  img = np.expand_dims(img, -1)
  return img

def __getitem__(self, idx):
  # select batch
  filenames = self.filenames[idx*self.batch_size : (idx + 1)*self.batch_size] # Image path

  if self.predict:
    # Load files for this batch
    imgs = [self.__loadpredict__(filename) for filename in filenames]
    imgs = np.array(imgs) # Create numpy arrays for batch images
    return imgs, filenames
  else:
    # Load files for this batch
    items = [self.__load__(filename) for filename in filenames]
    imgs, msks = zip(*items) # Output of __load__ is a tuple with imgs and masks, so uUnzip
    imgs = np.array(imgs)    # Create numpy arrays for batch images
    msks = np.array(msks)    # Create numpy arrays for batch masks
    return imgs, msks

def on_epoch_end(self):
  if self.shuffle:
    random.shuffle(self.filenames)

def __len__(self):
  if self.predict:
    return int(np.ceil(len(self.filenames) / self.batch_size))
  else:
    return int(len(self.filenames) / self.batch_size)
```

Models Used

- Model with Conv2D Layers
- Model with SeparableConv2D Layers
- Model with SeparableConv2D Layers additional layers
- Model with Resnet Blocks
- Model for Classification with SeparableConv2D Layers

Deciding Models and Model Building

- Model 1 with Conv2D Layers

```
# Model 1 with Conv2D Layers
model1 = Sequential()

# Add Convolution layers with 32 kernels of 3X3 shape with activation function ReLU
model1.add(Conv2D(4, (3, 3), input_shape = (IMAGE_SIZE, IMAGE_SIZE, 1), activation = 'relu', padding = 'same'))
model1.add(MaxPooling2D(pool_size = (2, 2))) # Max Pooling layer of size 2X2

model1.add(Conv2D(16, (2, 2), activation = 'relu', padding = 'same'))
model1.add(MaxPooling2D(pool_size = (2, 2)))
model1.add(BatchNormalization())

model1.add(Conv2D(64, (2, 2), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2, 2)))
model1.add(Dropout(0.4))

model1.add(Conv2D(128, (2, 2), activation = 'relu', padding = 'same'))
model1.add(MaxPooling2D(pool_size = (2, 2)))
model1.add(Dropout(0.3))

# 1 unit with Sigmoid activation for binary classification
model1.add(Conv2D(1, 1, activation='sigmoid'))

# UpSampling Layer to bring the image back to the input size
model1.add(UpSampling2D(16))

model1.summary()

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Training accuracy : 97.34%
Validation accuracy : 97.55%

Deciding Models and Model Building

- Model 2 with SeperableConv2D Layers

```
# Model 2 with SeperableConv2D Layers
model2 = Sequential() # Create and instance of Sequential Model

model2.add(Conv2D(filters=16, kernel_size=(2,2), input_shape=(IMAGE_SIZE, IMAGE_SIZE, 1), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size = (2, 2)))

model2.add(SeparableConv2D(filters=32, kernel_size=(2,2), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size = (2, 2)))

model2.add(SeparableConv2D(filters=64, kernel_size=(2,2), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size = (2, 2)))

model2.add(SeparableConv2D(filters=128, kernel_size=(2,2), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size = (2, 2)))

model2.add(SeparableConv2D(filters=256, kernel_size=(2,2), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size = (2, 2)))

model2.add(SeparableConv2D(filters=256, kernel_size=(2,2), activation='relu', padding='same'))
model2.add(BatchNormalization(momentum=0.9))
model2.add(MaxPooling2D(pool_size = (2, 2)))

model2.add(Conv2D(1, 1, activation='sigmoid'))

model2.add(UpSampling2D())
model2.add(UpSampling2D())
model2.add(UpSampling2D())
model2.add(UpSampling2D())
model2.add(UpSampling2D())
model2.add(UpSampling2D())

model2.summary()

# Compile model
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Training accuracy : 97.33%
Validation accuracy : 97.37%

Deciding Models and Model Building

- Model 3 with SeperableConv2D blocks with greater depth

```
# Model 3 with SeperableConv2D Layers
inputs = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1))

# First conv block
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Second conv block
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=32, 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=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Fourth conv block
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=128, 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)

# 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=0.2)(x)

# Sigmoid activation for binary classification
x = Conv2D(filters=1, kernel_size=(2,2), activation='sigmoid', padding='same')(x)

output = UpSampling2D(32)(x)

# Creating model
model3 = Model(inputs=inputs, outputs=output)

model3.summary()
```

Training accuracy : 96.21%
Validation accuracy : 96.01%

Deciding Models and Model Building

- Model 4 with ResNet blocks

```
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)
  # Added start
  #x = keras.loyers.Conv2D(channeLs, 1, padding.'same', use bias-FaLse)(x)
  #x = keras.loyers.MaxPooL20(2)(x)
  # Added End
  return x
```

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

  #Added Start
  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)
  #Added End

  addInput = x;
  print("Add input shape:", addInput.shape)
  print("Resnet block input shape:", inputs.shape)
  resBlockOut = keras.layers.add([addInput, inputs])
  print("Resnet block out shape:", resBlockOut.shape)
  out = keras.layers.concatenate([resBlockOut, addInput], axis=3)
  print("concat block out shape:", out.shape)
  out = keras.layers.Conv2D(channels, 1, padding='same', use_bias=False)(out)
  print("mixed block out shape:", out.shape)
  return out
```

Training accuracy : 96.57%
Validation accuracy : 95.48%

BCE + IOU as Loss
Mean IOU as Metric

```
# 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 0.5 * keras.losses.binary_crossentropy(y_true, y_pred) + 0.5 * iou_loss(y_true, y_pred)

# Mean IOU as a metric
def mean_iou(y_true, y_pred):
  y_pred = tf.round(y_pred)
  intersect = tf.reduce_sum(y_true * y_pred, axis=[1, 2, 3])
  union = tf.reduce_sum(y_true, axis=[1, 2, 3]) + tf.reduce_sum(y_pred, axis=[1, 2, 3])
  smooth = tf.ones(tf.shape(intersect))
  return tf.reduce_mean((intersect + smooth) / (union - intersect + smooth))
```

Model Training

```
[ ] # Define Callbacks
    checkpoint = ModelCheckpoint(filepath='best_weights.hdf5', save_best_only=True, save_weights_only=True)
    lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=1, verbose=2, mode='max')
    early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=1, mode='min')
```

```
[ ] # Generate training and validation datasets
    folder = 'stage_2_train_images/'
    train_gen = DataSequence(folder, train_filenames, pneumonia_locations, batch_size=BATCH_SIZE, image_size=IMAGE_SIZE, shuffle=False, augment=True, predict=False)
    valid_gen = DataSequence(folder, valid_filenames, pneumonia_locations, batch_size=BATCH_SIZE, image_size=IMAGE_SIZE, shuffle=False, predict=False)
```

```
[ ] # Model1
    history1 = model1.fit_generator(train_gen, validation_data=valid_gen, callbacks=[lr_reduce], epochs=2, shuffle=True, verbose=1)
```

```
Epoch 1/2
 30/755 [>.............................] - ETA: 3:54:07 - loss: 0.5515 - accuracy: 0.9611 - mean_iou: 0.6355/usr/local/lib/python3.6/dist-packages/keras/utils/data_uti
  UserWarning)
200/755 [======>......................] - ETA: 2:57:00 - loss: 0.5385 - accuracy: 0.9695 - mean_iou: 0.7040/usr/local/lib/python3.6/dist-packages/keras/utils/data_uti
  UserWarning)
253/755 [=========>...................] - ETA: 2:39:54 - loss: 0.5375 - accuracy: 0.9699 - mean_iou: 0.7098/usr/local/lib/python3.6/dist-packages/keras/utils/data_uti
  UserWarning)
584/755 [======================>.......] - ETA: 53:30 - loss: 0.5335 - accuracy: 0.9710 - mean_iou: 0.7155/usr/local/lib/python3.6/dist-packages/keras/utils/data_utils
  UserWarning)
755/755 [==============================] - 15352s 20s/step - loss: 0.5313 - accuracy: 0.9703 - mean_iou: 0.7049 - val_loss: 0.5768 - val_accuracy: 0.8882 - val_mean_io
Epoch 2/2
755/755 [==============================] - 3494s 5s/step - loss: 0.5224 - accuracy: 0.9671 - mean_iou: 0.6679 - val_loss: 0.5386 - val_accuracy: 0.9670 - val_mean_iou:

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00030000000142492354.
```

Classification Model

```
def __getitem__(self, idx):
  # select batch
  filenames = self.filenames[idx*self.batch_size : (idx + 1)*self.batch_size]

  if self.predict:
    # Load files for this batch
    imgs = [self.__loadpredict__(filename) for filename in filenames]
    imgs = np.array(imgs) # Create numpy arrays for batch images
    return imgs, filenames
  else:
    # Load files for this batch
    items = [self.__load__(filename) for filename in filenames]
    imgs, tgts = zip(*items) # Output of __load__ is a tuple with imgs and targets, so Unzip the images and targets
    imgs = np.array(imgs)    # Create numpy arrays for batch images
    tgts = np.array(tgts)    # Create numpy arrays for target
    return imgs, tgts
```

Classification Model

```
[ ] inputs = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1))

    # First conv block
    x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
    x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = MaxPool2D(pool_size=(2, 2))(x)

    # Second conv block
    x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = SeparableConv2D(filters=32, 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=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPool2D(pool_size=(2, 2))(x)

    # Fourth conv block
    x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = SeparableConv2D(filters=128, 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)

    # 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=0.2)(x)
```

Classification Model

```
# FC layer
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(rate=0.7)(x)
x = Dense(units=128, activation='relu')(x)
x = Dropout(rate=0.5)(x)
x = Dense(units=64, activation='relu')(x)
x = Dropout(rate=0.3)(x)

# Output layer
output = Dense(units=1, activation='sigmoid')(x)
```

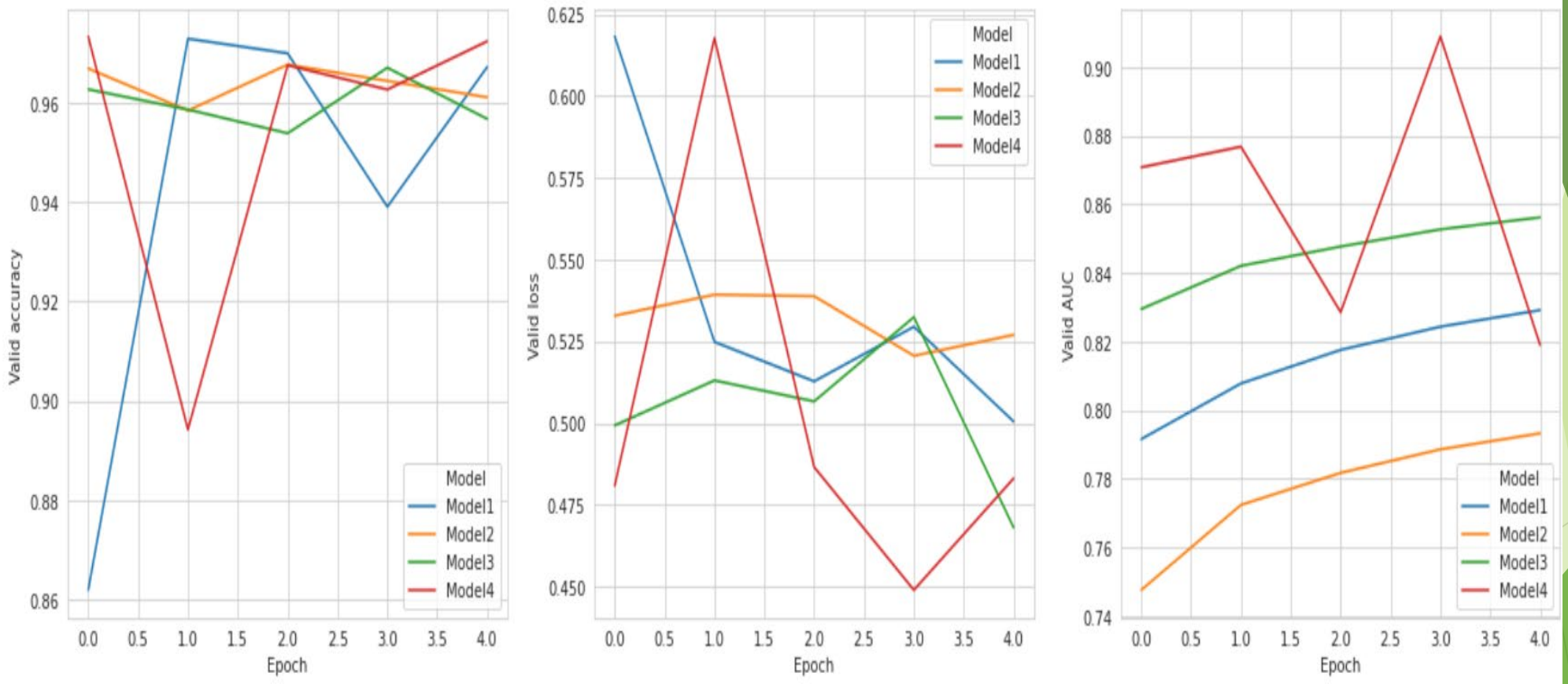
Training accuracy : 81.97%
Validation accuracy : 80.69%

Model Comparison

Model	Training accuracy	Validation accuracy	AUC
Model 1: Conv2D Layers	97.34%	97.55%	82.92
Model 2: SeperableConv2D Layers	97.33%	97.37%	79.32
Model 3: SeperableConv2D blocks with greater depth	96.21%	96.01%	85.62
Model 4: ResNet blocks	96.57%	95.48%	81.90
Classification Model	81.97%	80.69%	-

Model Comparison

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f479a0469e8>
```



Step by Step Approach

- Pre-processing by splitting training data into batches of size 32
- Created a basic model with only Conv2D layers
- To reduce the training parameters, created model with SeparableConv2D
- Built 4 models for predicting bounding boxes and evaluated based on Accuracy metric
- Built Classification model for predicting the presence of Pneumonia
- Picked model 3, SeperableConv2D blocks with greater depth, based on accuracy and loss.
- Verified actual and predicted bounding boxes using 1 batch(32) of validation samples
- Created submission file

Model Evaluation

- Based on AUC, Accuracy and Loss, selected model 3, SeperableConv2D blocks

```python
# REGRESSION MODEL TO PREDICT BBOX COORDINATES
inputs = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1))

# First conv block
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Second conv block
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=32, 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=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Fourth conv block
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=128, 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)

# 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=0.2)(x)

# Sigmoid activation for binary classification
x = Conv2D(filters=1, kernel_size=(2,2), activation='sigmoid', padding='same')(x)

output = UpSampling2D(32)(x)

# Creating model
model3 = Model(inputs=inputs, outputs=output)
model3.summary()
```

Model Evaluation

- Model Structure for Model 3
  - 5 SeparableConv2D blocks with MaxPooling and BatchNormalization layers
  - Conv2D Sigmoid Activation
  - 32 Upsampling Layers
  - Optimzer = Adam
  - Loss = iou_bce_loss
  - Metrics = accuracy
  - Epochs = 8

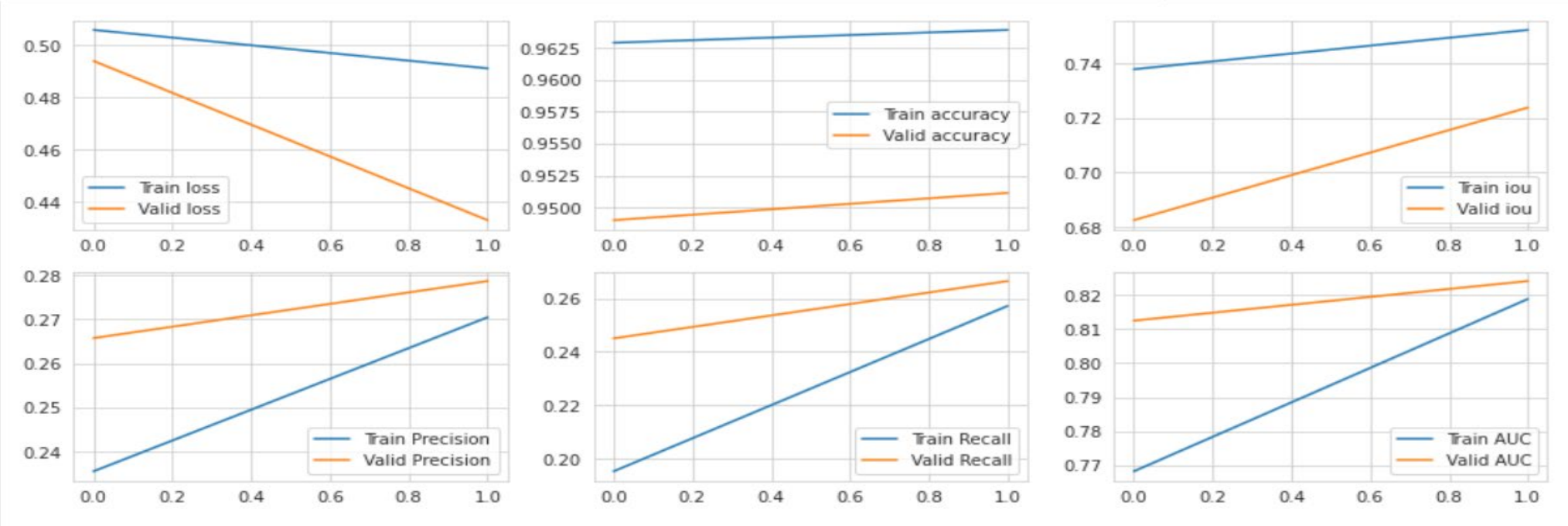
Model Evaluation

```
# Use Adam optimizer, combination of BCE and IOU loss, and monitor model performance using accuracy and average IOU
model3.compile(optimizer='adam', loss=iou_bce_loss, metrics=['accuracy', mean_iou, tf.keras.metrics.Precision(), tf.keras.metrics.Recall(), tf.keras.metrics.AUC()])
```

```
[ ]  # Model3
     #history3 = model3.fit_generator(train_gen, validation_data=valid_gen, epochs=2, callbacks=[checkpoint, lr_reduce], verbose=1, shuffle=True)
```

```
Epoch 1/2
755/755 [==============================] - 5751s 8s/step - loss: 0.5060 - accuracy: 0.9629 - mean_iou: 0.7378 - precision_1: 0.2355 - recall_1: 0.1952 - auc_1: 0.7682 - val_loss: 0.4941 - val_accuracy:
Epoch 2/2
755/755 [==============================] - 5684s 8s/step - loss: 0.4912 - accuracy: 0.9639 - mean_iou: 0.7523 - precision_1: 0.2705 - recall_1: 0.2572 - auc_1: 0.8189 - val_loss: 0.4330 - val_accuracy:

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00030000000142492354.
```



Model Evaluation

- Based on accuracy and loss, selected model 3, SeperableConv2D blocks

```
df_regr_metrics = pd.DataFrame(columns=['Model','Epoch','Train loss','Valid loss','Train accuracy','Valid accuracy','Train iou','Valid iou','Train Precision','Valid Precision','
for i, epoch_num in enumerate(history.epoch):
  df_regr_metrics = df_regr_metrics.append({
                    'Model':'Model3',
                    'Epoch': int(epoch_num),
                    'Train loss':history.history["loss"][i],
                    'Valid loss':history.history["val_loss"][i],
                    'Train accuracy':history.history["accuracy"][i],
                    'Valid accuracy':history.history["val_accuracy"][i],
                    'Train iou':history.history["mean_iou"][i],
                    'Valid iou':history.history["val_mean_iou"][i],
                    'Train Precision':history.history['precision_1'][i],
                    'Valid Precision':history.history["val_precision_1"][i],
                    'Train Recall':history.history['recall_1'][i],
                    'Valid Recall':history.history["val_recall_1"][i],
                    'Train AUC':history.history['auc_1'][i],
                    'Valid AUC':history.history["val_auc_1"][i]}, ignore_index=True)
df_regr_metrics
```

	Model	Epoch	Train loss	Valid loss	Train accuracy	Valid accuracy	Train iou	Valid iou	Train Precision	Valid Precision	Train Recall	Valid Recall	Train AUC	Valid AUC
0	Model3	0	0.506007	0.494073	0.962898	0.949001	0.737834	0.682489	0.235494	0.265730	0.19519	0.245055	0.768172	0.812514
1	Model3	1	0.491239	0.432990	0.963905	0.951135	0.752271	0.723777	0.270460	0.278685	0.25725	0.266564	0.818901	0.824171

Submission File Creation

```
submission_dict = {}
# Predict the bbox coordinates for the test dataset and
for imgs, filenames in test_gen:
  # predict batch of images
  preds = model.predict(imgs)
  print(filenames.shape)
  print(preds.shape)

  for filename, pred in zip(filenames, preds):
    # resize predicted mask
    pred = resize(pred, (1024, 1024), mode='reflect')
    # threshold predicted mask
    comp = pred[:, :, 0] > 0.5
    # apply connected components
    comp = measure.label(comp)
    # apply bounding boxes
    predictionString = ''
    for region in measure.regionprops(comp):
      # retrieve x, y, height and width
      y, x, y2, x2 = region.bbox
      height = y2 - y
      width = x2 - x
      # proxy for confidence score
      conf = np.mean(pred[y:y+height, x:x+width])
      predictionString += str(conf) + ' ' + str(x) + ' ' + str(y) + ' ' + str(width) + ' ' + str(height)
    filename = filename.split('.')[0]
    submission_dict[filename] = predictionString
  # stop if we've got them all
  if len(submission_dict) >= len(test_filenames):
    break
      #sumbission_df = sumbission_df.append({'patientId':patientId, 'PredictionString': predictionString}, ignore_index=True)

# save dictionary as csv file
sub = pd.DataFrame.from_dict(submission_dict,orient='index')
sub.index.names = ['patientId']
sub.columns = ['PredictionString']
sub.to_csv('submission.csv')
```

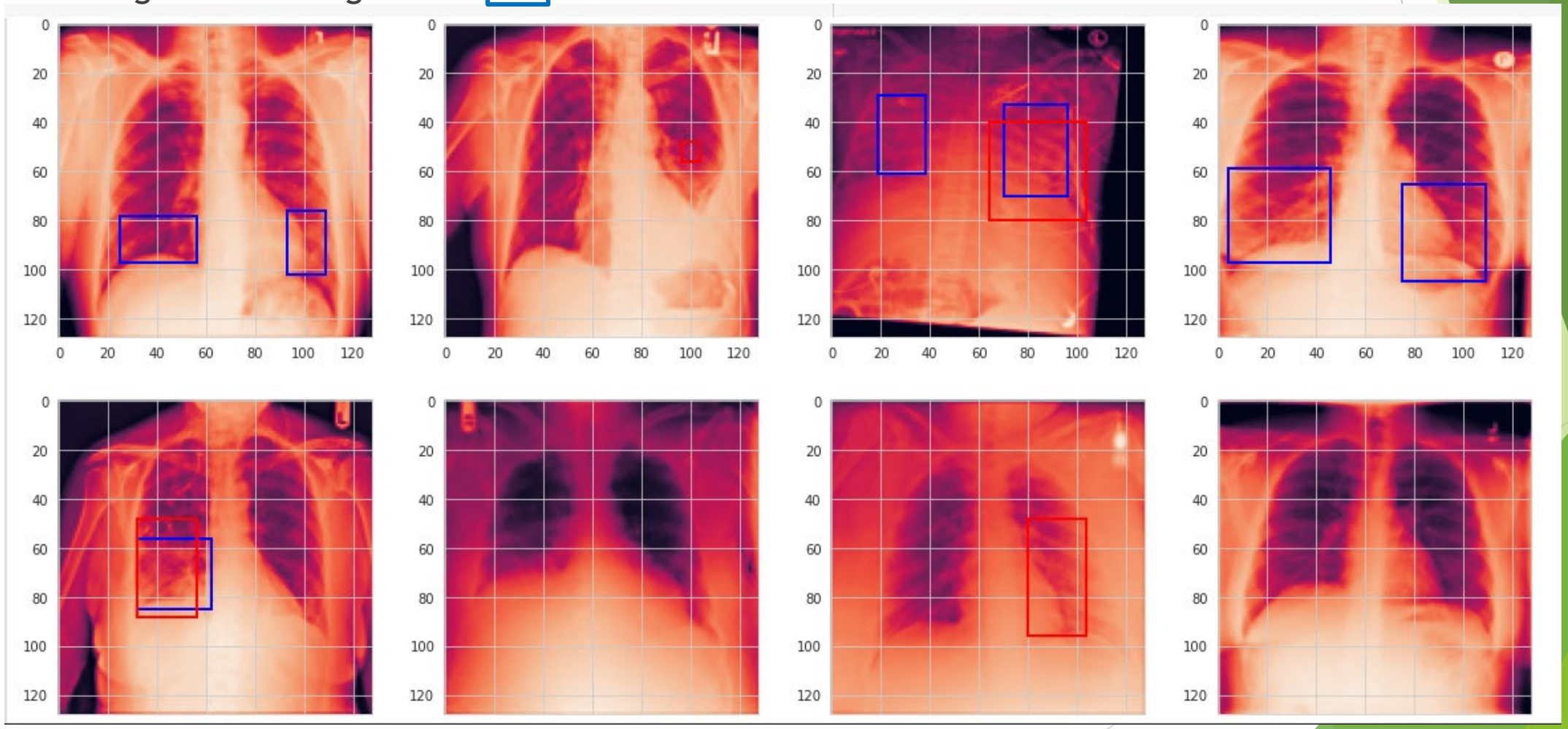
```
sub.to_csv('/content/drive/My Drive/MainProject/submission.csv')
```

Submission File

patientId	PredictionString
24fe325a-be98-405e-bbf5-cbe64aa82e5c	
25f09d92-e440-4e16-aca2-5baca1a76c89	
26636455-c98d-49a1-8b48-7025f535f982	0.77659607 508 257 259 510
255354f6-9379-493a-90d1-5dd36ac660bc	
25c98e9d-637f-4e6f-8ad3-f9f85a7f7590	
25d2e423-40f3-4c47-becd-e05908e3ced4	
264d4100-0a4a-460d-81fd-e781bbf6ca86	
266490ca-ce52-4f5b-92ff-082dae7967c0	
26903582-dbc1-4427-b9da-45e2fa00ed17	
24e04bef-a0c2-439e-9181-5fa26c162fa4	0.5576752 259 258 507 507
2634cbd2-0a3f-4503-bc07-b8f4f2da80c4	
2557ee07-5fc1-4eba-915e-f52d3d898f1e	0.7287349 257 257 511 511
25840be8-f6cf-4df1-893b-80412b99a417	0.52446324 515 260 249 504
2657b0a9-63ec-43b2-810e-b665f879f309	
2568153b-bdde-4292-8968-47f8e74cd400	
26a937bc-a3e4-41a1-9a1a-a63506d4b49d	0.8797624 256 256 512 512
265dd221-9049-4bca-b5c0-4118dafa55c5	
268116a6-2304-4316-b5b3-3073fc5467b1	
257e34f1-f461-45fd-ab15-ec4cbfada52a	
2578fa2d-6cf3-41d1-b233-89587b1126d8	
2676fc9d-7ace-4896-b698-17fc68131851	0.6088609 258 258 255 507
258f5240-8010-4472-91fe-494f75d59a46	0.56559247 513 258 252 506

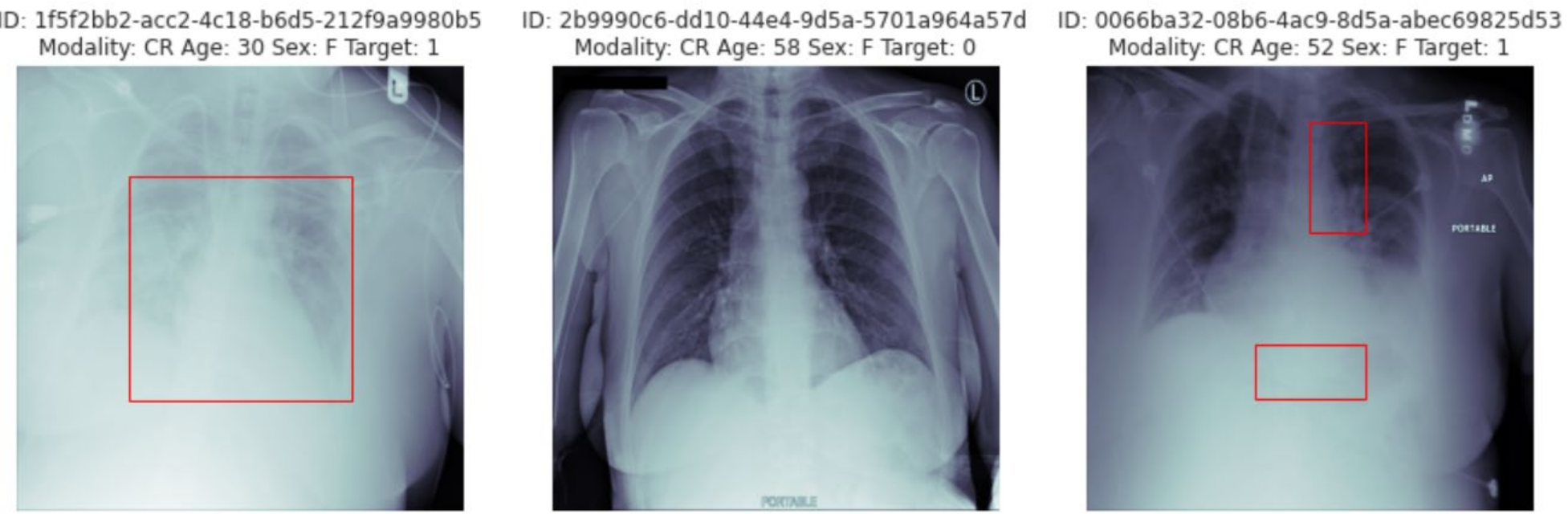
Model Output for Training Data(Model 3)

- Predicted bounding box
- Original bounding box



Model Output for Test Data(Model 3)

- Predicted bounding box



As we can see, the model is predicting correctly.

- For the first patient, the bounding box is bigger as the whole chest looks congested
- For the second patient, we dont have a bounding box, as the lungs appear to be clear
- For the third patient, pneumonia is detected in 2 different areas marked by the two seperate bounding boxes.

Conclusion

- Our model is able to predict the bounding boxes around Pneumonic patches w ith accuracy of 87% on test data.
- Out of 3000 patients in the test set, 935 appear to have pneumonia for whom one or more bounding boxes are predicted by our model.
- Approximately 31% of patients are predicted to have pneumonia which is cons istent with the training set data.
- This project uses recent techniques in the field of computer vision and deep l earning and visualizations of key outcomes.

Implications

- Application can predict if the patient is affected with Pneumonia
- It pinpoints the affected areas of lungs using bounding boxes and the confidence levels associated with each of them
- This will help the doctors by reducing the number of CXRs they have to look through
- This project will help health care professionals by making quick and accurate predictions of pneumonia in CXR images.

Limitations

- Currently, input images size is fixed. This can be modified to accept images of different sizes.

How to improve model performance?

- Image Augmentation to increase the number of images where Target=1, which is our class of interest
- Increase the number of training epochs
- Choose models based on performance

Closing Reflections

- Learnings
  - Use of pydicom package to view .dcm medical images
  - Preprocessing of image related data
  - Use of custom loss and metric functions
  - Resnet and SeperableConv2D Layers
  - Collaboration between team members for different tasks
  - Understanding of complexity of real-world data and projects
- Possible Improvements
  - Use a pretrained model such as YOLO or Faster-RCNN as benchmark
  - Image Augmentation

Thank you!