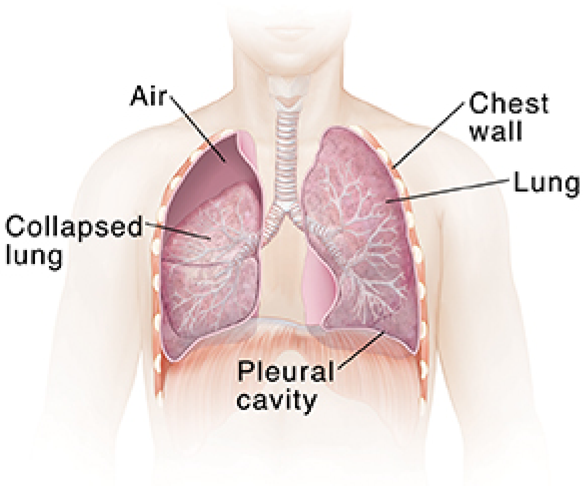


SIIM-ACR Pneumothorax Segmentation

1. Business Problem

1.1 Description

A pneumothorax or collapsed lungs is a medical condition that is responsible for making the people suddenly gasp for air, and feel helplessly breathless for no apparent reason. It can be a complete lung collapse or a collapse of a portion of the lung. It is usually diagnosed by a radiologist with several years of experience on a chest x- ray; which sometimes is very difficult to confirm. Our goal is to classify(if present segment) pneumothorax from a set of chest radiographic images.

This is a Kaggle problem. (https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/overview)

1.2 Problem Statement

Classify pneumothorax from a set of chest radiographic images and if present segment the regions of pneumnothorax.

1.3 Sources

- Source : https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/overview
-

1.4 Real world/Business Objectives and Constraints

- Classify pneumothorax and if present segment it.
- Maximize the overlap between the actual mask and predicted mask(Dice).
- No such latency concerns.

2. Machine Learning Problem

2.1 Data

2.1.1 Data Overview

- Data is present in under folder SIIM. It contains three folders and files,

- dicom-images-test
- dicom-images-train
- train-rle.csv

Note : I downloaded the data from https://www.kaggle.com/seesee/siim-train-test as the data is removed from the Cloud Healthcare API.

2.2 Mapping the real world problem to a Deep Learning Problem

2.2.1 Type of Deep Learning Problem

It is a basically a semantic image segmentation problem, where we have to segment areas of pneumothorax.

2.2.2 Performance Metric

Source: https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/data

Metric:

- Focal Loss + Dice Loss
    - Focal loss is very good for imbalanced data as it focuses more on hard examples than easy examples. Our data is very imbalanced as the area having pneumothorax is very small. Dice is best metric overall. So we are going for the combination of the two.

2.2.3 Necessary Imports

```
import numpy as np

def mask2rle(img, width, height):
    rle = []
    lastColor = 0;
    currentPixel = 0;
    runStart = -1;
    runLength = 0;

    for x in range(width):
        for y in range(height):
            currentColor = img[x][y]
            if currentColor != lastColor:
                if currentColor == 255:
                    runStart = currentPixel;
                    runLength = 1;
                else:
                    rle.append(str(runStart));
                    rle.append(str(runLength));
                    runStart = -1;
                    runLength = 0;
                    currentPixel = 0;
            elif runStart > -1:
                runLength += 1
            lastColor = currentColor;
            currentPixel+=1;

    return " ".join(rle)

def rle2mask(rle, width, height):
    mask= np.zeros(width* height)
    array = np.asarray([int(x) for x in rle.split()])
    starts = array[0::2]
    lengths = array[1::2]

    current_position = 0
```

```
    for index, start in enumerate(starts):
        current_position += start
        mask[current_position:current_position+lengths[index]] = 255
        current_position += lengths[index]

    return mask.reshape(width, height)
```

```
!pip install pydicom
```

```
Collecting pydicom
  Downloading https://files.pythonhosted.org/packages/f4/15/df16546bc59bfca390cf072d473
     |████████████████████████████████| 1.9MB 6.2MB/s
Installing collected packages: pydicom
Successfully installed pydicom-2.1.2
```

```
import pydicom
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import glob
from tqdm import tqdm
import seaborn as sns
```

```
import warnings
warnings.filterwarnings("ignore")
import gc
gc.collect()
import cv2
import tensorflow as tf
import keras
from PIL import Image, ImageDraw
from PIL import ImagePath
import imgaug.augmenters as iaa
from skimage import exposure
```

1. Creating train and test dataset

```
data_rle_train = pd.read_csv("./siim/train-rle.csv")
data_rle_train.head()
```

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...

```
#The second column contains a space infront of it so manually rewriting it
data_rle_train.columns = ['ImageId', 'EncodedPixels']


#https://www.kaggle.com/jesperdramsch/intro-chest-xray-dicom-viz-u-nets-full-data
def getInfoDICOM(path, toPrint = False, train = True):

    info = {}

    path = os.path.join(path)
    #reading the data using methods of pydicom
    data = pydicom.dcmread(path)
    info['path'] = path
    info['age'] = data.PatientAge
    info['sex']= data.PatientSex
    info['ImageId'] = data.SOPInstanceUID

    if train: #test doesn't have encoded pixels data, we have to predict them
        #SOPInstanceUID contains the storage type which resembles the image ids given in trai
        encodedPixels = data_rle_train[data_rle_train['ImageId'] == data.SOPInstanceUID]["Enc
        info['encodedPixels'] = encodedPixels

        info['lenOfEncodedPixels'] = len(encodedPixels)

        #this is for visualization purpose
        if '-1' in encodedPixels or len(encodedPixels) == 0:
            info['has_pneumothorax'] = 0
        else:
            info['has_pneumothorax'] = 1

    if toPrint:
        print("Path...............:", path)
        print("Patient's Name.....:", data.PatientName)
        print("Patient's Id.......:", data.PatientID)
        print("Patient's Age......:", data.PatientAge)
        print("Patient's Sex......:", data.PatientSex)

        print("Original X Ray")
        plt.figure(figsize=(10,10))
        plt.imshow(data.pixel_array, cmap=plt.cm.bone)
        plt.show()


    return info
```

```
#creation of train dataset
#https://stackoverflow.com/questions/33747968/getting-file-list-using-glob-in-python
filesList = glob.glob('./siim/dicom-images-train/*/*/*.dcm')
train = pd.DataFrame()
train_list = []
for file in tqdm(filesList):
    info = getInfoDICOM(file, False, True)
    train_list.append(info)
train = pd.DataFrame(train_list)
```

```
100%|██████████| 12089/12089 [00:40<00:00, 296.59it/s]
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12089 entries, 0 to 12088
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   path                12089 non-null  object
 1   age                 12089 non-null  object
 2   sex                 12089 non-null  object
 3   ImageId             12089 non-null  object
 4   encodedPixels       12089 non-null  object
 5   lenOfEncodedPixels  12089 non-null  int64 
 6   has_pneumothorax    12089 non-null  int64 
dtypes: int64(2), object(5)
memory usage: 661.2+ KB
```

```
train.head(5)
```

	path	age	sex	ImageId	enco
0	./siim/dicom-images-train/1.2.276.0.7230010.3....	73	M	1.2.276.0.7230010.3.1.4.8323329.6277.151787519...	
1	./siim/dicom-images-train/1.2.276.0.7230010.3....	10	F	1.2.276.0.7230010.3.1.4.8323329.7035.151787520...	
2	./siim/dicom-images-train/1.2.276.0.7230010.3....	64	M	1.2.276.0.7230010.3.1.4.8323329.14139.15178752...	[978 20 33
3	./siim/dicom-images-train/1.2.276.0.7230010.3....	22	F	1.2.276.0.7230010.3.1.4.8323329.3216.151787517...	
					[256

```
#creation of test dataset
filesList = glob.glob('./siim/dicom-images-test/*/*/*.dcm')
test = pd.DataFrame()
test_list = []
```

```
for file in tqdm(filesList):
    info = getInfoDICOM(file, False, False)
    test_list.append(info)
test = pd.DataFrame(test_list)
```

```
100%|██████████| 3205/3205 [00:12<00:00, 262.88it/s]
```

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3205 entries, 0 to 3204
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   path     3205 non-null   object
 1   age      3205 non-null   object
 2   sex      3205 non-null   object
 3   ImageId  3205 non-null   object
dtypes: object(4)
memory usage: 100.3+ KB
```

```
test.head(5)
```

	path	age	sex	ImageId
0	./siim/dicom-images-test/_/_/ID_af8eb43f4.dcm	22	M	ID_af8eb43f4
1	./siim/dicom-images-test/_/_/ID_114609bb6.dcm	19	M	ID_114609bb6
2	./siim/dicom-images-test/_/_/ID_2e2105281.dcm	41	M	ID_2e2105281
3	./siim/dicom-images-test/_/_/ID_e7eac7dab.dcm	41	M	ID_e7eac7dab
4	./siim/dicom-images-test/_/_/ID_7aef3400f.dcm	83	F	ID_7aef3400f

```
print("Columns having null values in train", train.columns[train.isnull().any()].tolist())
```

```
Columns having null values in train []
```

```
print("No of images having no mask", train[train['lenOfEncodedPixels'] == 0].shape[0])
```

```
No of images having no mask 42
```

```
train.loc[train.lenOfEncodedPixels == 0 , 'encodedPixels'] = np.array(-1)
```

2. Data Preparation

```
def computeMasks(train, path):
```

```
    mask_paths = []
    for index, row in tqdm(train.iterrows()):
        pixels = row['encodedPixels']
        mask = np.zeros((1024, 1024))
        if row['has_pneumothorax'] == 1:
            for pix in pixels:
                mask = mask + rle2mask(pix, 1024, 1024).T
        img = Image.fromarray(mask).convert('L')
        path2save = path + str(index+1) + ".jpg"
        img.save(path2save)
        mask_paths.append(path2save)
    train["mask"] = mask_paths
    return train


path = "siim/dicom-mask-train/"
try:
    os.makedirs(path)
except:
    pass
train = computeMasks(train, path)
train.head(2)
```

```
12089it [06:14, 32.28it/s]
```

	path	age	sex	ImageId	enco
0	./siim/dicom-images-train/1.2.276.0.7230010.3....	57	F	1.2.276.0.7230010.3.1.4.8323329.13904.15178752...	
1	./siim/dicom-images-train/1.2.276.0.7230010.3....	52	F	1.2.276.0.7230010.3.1.4.8323329.5535.151787518...	

```
def convertImagesToJpeg(train, path):
    images_paths = []
    for index, row in tqdm(train.iterrows()):
        path2dicom = row['path']
        ds = pydicom.read_file(path2dicom) # read dicom image
        img = ds.pixel_array # get image array
        img_mem = Image.fromarray(img) # Creates an image memory from an object exporting the
        path2save = path + str(index + 1)  + ".jpg"
        img_mem.save(path2save)
        images_paths.append(path2save)
    train["images_paths"] = images_paths
    return train


path = "siim/dicom-images-train-jpg/"
try:
    os.makedirs(path)
except:
```

```
    pass
train = convertImagesToJpeg(train, path)
train.head(2)
```

12089it [06:00, 33.51it/s]

	path	age	sex	ImageId	enco
0	./siim/dicom-images-train/1.2.276.0.7230010.3....	57	F	1.2.276.0.7230010.3.1.4.8323329.13904.15178752...	
1	./siim/dicom-images-train/1.2.276.0.7230010.3....	52	F	1.2.276.0.7230010.3.1.4.8323329.5535.151787518...	

```
path = "siim/dicom-images-test-jpg/"
try:
    os.makedirs(path)
except:
    pass
test = convertImagesToJpeg(test, path)
test.head(2)
```

3205it [01:32, 34.79it/s]

	path	age	sex	ImageId	images_paths
0	./siim/dicom-images-test/_/_/ID_bf9328240.dcm	38	F	ID_bf9328240	siim/dicom-images-test-jpg/1.jpg
1	./siim/dicom-images-	53	M	ID_a49ab23fd	siim/dicom-images-test-

3. Data Pipeline

```
import tensorflow as tf
# tf.compat.v1.enable_eager_execution()
from tensorflow import keras
from tensorflow.keras.layers import *
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import UpSampling2D
from tensorflow.keras.layers import MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Multiply
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormal
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.initializers import glorot_uniform, he_normal
from tensorflow.keras.losses import binary_crossentropy
```

```
from tensorflow.keras.losses import binary_crossentropy
K.set_image_data_format('channels_last')
```

```
fileNames = train['images_paths']
labels = train['has_pneumothorax']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(fileNames, labels, test_size=0.10, strati
```

```
X_train.head(2)
```

```
8489    siim/dicom-images-train-jpg/8490.jpg
9227    siim/dicom-images-train-jpg/9228.jpg
Name: images_paths, dtype: object
```

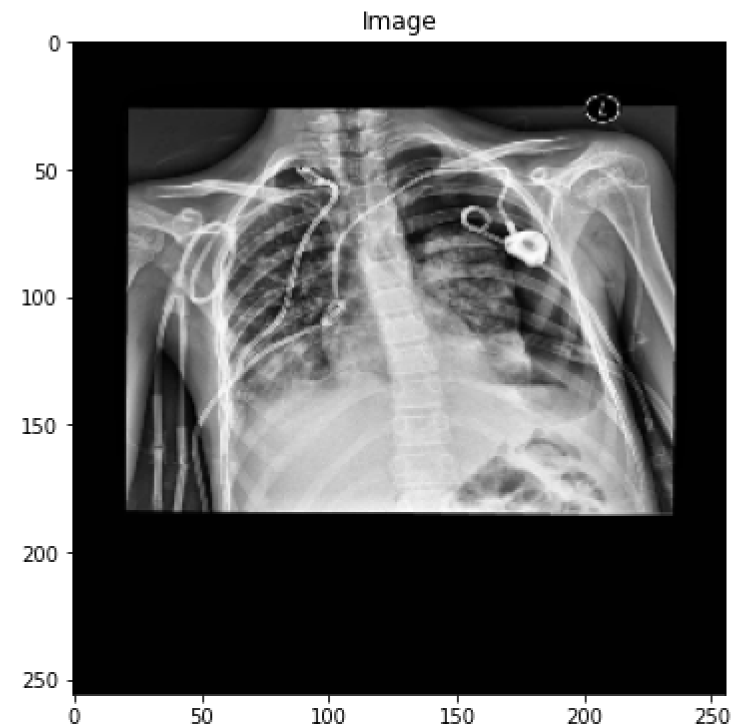
```
y_train.head(2)
```

```
8489    1
9227    0
Name: has_pneumothorax, dtype: int64
```

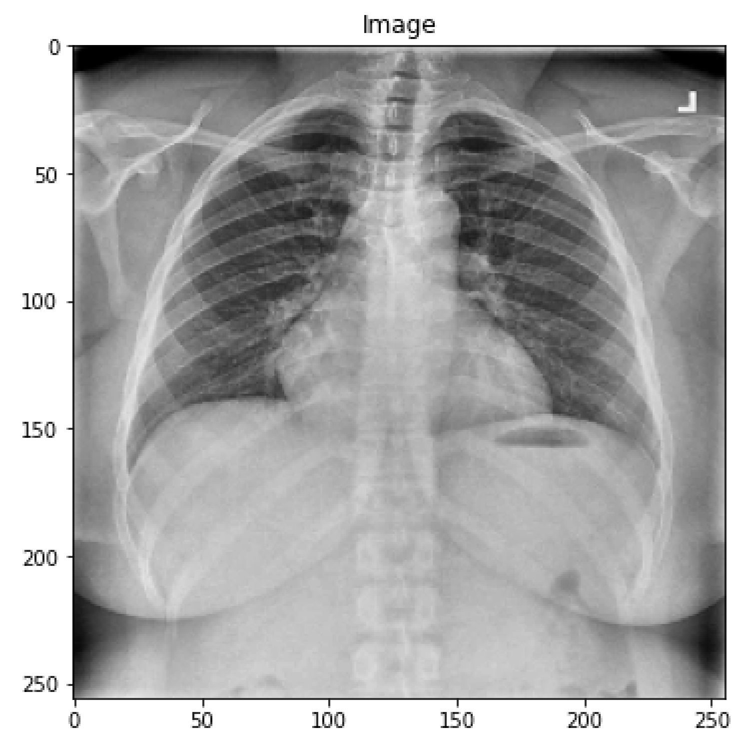
```
img_size = 256
```

```
def read_image(datapoint):
    image = tf.keras.preprocessing.image.load_img(datapoint)
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)     # contrast correction
    return image

def read_label(datapoint):
    datapoint = tf.reshape(datapoint, [1])
    return datapoint

def preprocess(image, label):
    def f(image, label):

        image = image.decode()
        image = read_image(image)

        a = np.random.uniform()
        if a>=0.80:
            b = np.random.uniform()
            if b<0.50:
                image = tf.image.flip_left_right(image)
            else:
                image = tf.image.flip_up_down(image)
```

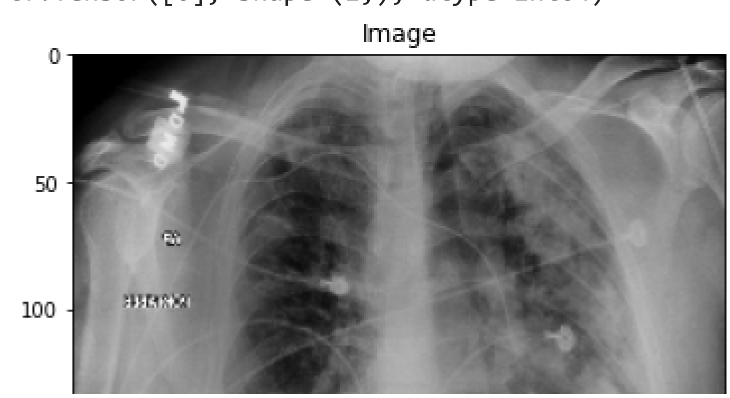
```
            label = read_label(label)
            return image, label
        images, labels = tf.numpy_function(f, [image, label], [tf.float32, tf.int64])
        images.set_shape([img_size, img_size, 3])
        labels.set_shape([1])
        return images, labels

    def tf_dataset(x, y, batch=8):
        dataset = tf.data.Dataset.from_tensor_slices((x, y))
        dataset = dataset.shuffle(buffer_size=15000)
        dataset = dataset.map(preprocess)
        dataset = dataset.batch(batch)
        dataset = dataset.prefetch(2)
        return dataset


    train_data = tf_dataset(X_train.values, y_train.values)


    for image, result in train_data.take(1):
        sample_image, sample_result = image, result
    for i in range(4):
        plt.figure(figsize=(6, 6))
        plt.title("Image")
        plt.imshow(tf.keras.preprocessing.image.array_to_img(sample_image[i]))
        plt.show()
        plt.close()
        print(sample_result[i])
```

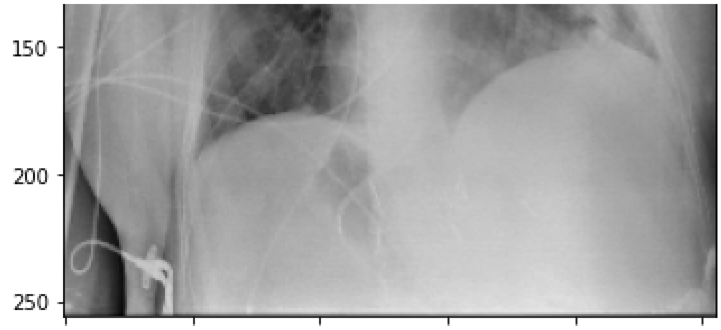



```
tf.Tensor([1], shape=(1,), dtype=int64)
```




```
tf.Tensor([0], shape=(1,), dtype=int64)
```






```
def read_image(datapoint):
    image = tf.keras.preprocessing.image.load_img(datapoint)
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)     # contrast correction
    return image

def read_label(datapoint):
    datapoint = tf.reshape(datapoint, [1])
    return datapoint

def preprocess(image, label):
    def f(image, label):

        image = image.decode()
        image = read_image(image)

        label = read_label(label)
        return image, label
    images, labels = tf.numpy_function(f, [image, label], [tf.float32, tf.int64])
    images.set_shape([img_size, img_size, 3])
    labels.set_shape([1])
    return images, labels

def tf_dataset(x, y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=15000)
    dataset = dataset.map(preprocess)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(2)
    return dataset


test_data = tf_dataset(X_test.values, y_test.values)
```

6. Model

```
from sklearn import metrics
```

```
class BasicBlock(tf.keras.layers.Layer):

    def __init__(self, filters, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1_basic = Conv2D(filters=filters, kernel_size=(3, 3),strides=stride, padding
        self.batchNorm1_basic = BatchNormalization()
        self.conv2_basic = Conv2D(filters=filters,kernel_size=(3, 3),strides=1,padding="same"
        self.batchNorm2_basic = BatchNormalization()
        if stride != 1:
            self.downsample_basic = tf.keras.Sequential()
            self.downsample_basic.add(Conv2D(filters=filters, kernel_size=(1, 1), strides=str
            self.downsample_basic.add(BatchNormalization())
        else:
            self.downsample_basic = lambda x: x

    def call(self, inputs, training=None, **kwargs):
        residual = self.downsample_basic(inputs)

        x = self.conv1_basic(inputs)
        x = self.batchNorm1_basic(x, training=training)
        x = tf.nn.relu(x)
        x = self.conv2_basic(x)
        x = self.batchNorm2_basic(x, training=training)

        output = tf.nn.relu(tf.keras.layers.add([residual, x]))

        return output

def make_basic_block_layer(filter_num, blocks, stride=1):
    res_block = tf.keras.Sequential()
    res_block.add(BasicBlock(filter_num, stride=stride))

    for _ in range(1, blocks):
        res_block.add(BasicBlock(filter_num, stride=1))

    return res_block



class Resnet34(tf.keras.Model):
    def __init__(self, input_shape, layer_params):
        super().__init__()

        self.conv1_resnet = tf.keras.layers.Conv2D(filters=32,
                                                   kernel_size=(7, 7),
                                                   strides=2,
                                                   padding="same")
        self.bn1_resnet = tf.keras.layers.BatchNormalization()
        self.pool1_resnet = tf.keras.layers.MaxPool2D(pool_size=(3, 3),
                                                      strides=2,
                                                      padding="same")
```

```
        self.layer1_resnet = make_basic_block_layer(filter_num=32,
                                                     blocks=layer_params[0])
        self.layer2_resnet = make_basic_block_layer(filter_num=64,
                                                     blocks=layer_params[1],
                                                     stride=2)
        self.layer3_resnet = make_basic_block_layer(filter_num=128,
                                                     blocks=layer_params[2],
                                                     stride=2)
        self.layer4_resnet = make_basic_block_layer(filter_num=256,
                                                     blocks=layer_params[3],
                                                     stride=2)

        self.avgpool_resnet = tf.keras.layers.AveragePooling2D()
        self.flatten = tf.keras.layers.Flatten()

        self.dense1 = tf.keras.layers.Dense(300, activation = 'relu')
        self.dense2 = tf.keras.layers.Dense(50, activation = 'relu')
        self.final = tf.keras.layers.Dense(1, activation= 'sigmoid')


    def call(self, inputs, training=None, mask=None):
        x = self.conv1_resnet(inputs)
        x = self.bn1_resnet(x, training=training)
        x = tf.nn.relu(x)
        x = self.pool1_resnet(x)
        x = self.layer1_resnet(x, training=training)
        x = self.layer2_resnet(x, training=training)
        x = self.layer3_resnet(x, training=training)
        x = self.layer4_resnet(x, training=training)
        x = self.avgpool_resnet(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        output = self.final(x)
        return output


model3 = Resnet34((256, 256, 3),[3, 4, 6, 3])


class History_Callback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs={}):

        val_predict = []
        for i in range(len(X_test)):
            image = tf.keras.preprocessing.image.load_img(X_test.iloc[i])
            image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
            image = tf.image.resize(image, [img_size, img_size])
            image = image / 255.0
            image = exposure.equalize_adapthist(image)
```

```
            image = tf.expand_dims(image, axis = 0)
            predict = self.model.predict(image)
            val_predict.append(np.squeeze(predict, axis = -1).round())

        val_targ = y_test
        f1_score = metrics.f1_score(val_targ, val_predict)
        auc = metrics.roc_auc_score(val_targ, val_predict)
        recall = metrics.recall_score(val_targ, val_predict)

        print("recall:  {recall} - f1Score: {f1score} - AUC: {AUC}".format(f1score = f1_score


history_Model1 = History_Callback()


callbacks = [
    tf.keras.callbacks.ModelCheckpoint('./best_model3.h5', save_weights_only=True, save_best_
                                       mode='min'),
             history_Model1
]
optimizer = tf.keras.optimizers.Adam(0.001)


model3.compile(optimizer, loss = 'binary_crossentropy' , metrics=['accuracy'])


history3 = model3.fit(train_data, steps_per_epoch=len(train_data), epochs=25,\
                              validation_data=test_data,callbacks=callbacks, )
```

```
1360/1360 [==============================] - 613s 451ms/step - loss: 0.5160 - accuracy
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 7/25
1360/1360 [==============================] - 642s 472ms/step - loss: 0.5149 - accuracy
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 8/25
1360/1360 [==============================] - 640s 471ms/step - loss: 0.5180 - accuracy
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 9/25

1360/1360 [==============================] - 648s 477ms/step - loss: 0.4952 - accuracy
recall:  0.5543071161048689 - f1Score: 0.33789954337899547 - AUC: 0.5324614136787614
Epoch 10/25
1360/1360 [==============================] - 652s 479ms/step - loss: 0.4936 - accuracy
recall:  0.0449438202247191 - f1Score: 0.07894736842105263 - AUC: 0.5092022710465421
Epoch 11/25
1360/1360 [==============================] - 643s 473ms/step - loss: 0.4767 - accuracy
recall:  1.0 - f1Score: 0.3617886178861789 - AUC: 0.5
Epoch 12/25
1360/1360 [==============================] - 640s 471ms/step - loss: 0.4812 - accuracy
recall:  0.0 - f1Score: 0.0 - AUC: 0.4994692144373673
Epoch 13/25
1360/1360 [==============================] - 641s 472ms/step - loss: 0.4668 - accuracy
recall:  0.08239700374531835 - f1Score: 0.1423948220064725 - AUC: 0.5305827906200052
Epoch 14/25
1360/1360 [==============================] - 647s 476ms/step - loss: 0.4446 - accuracy
recall:  0.6292134831460674 - f1Score: 0.4132841328413284 - AUC: 0.6139697988978745
```

```
Epoch 15/25
1360/1360 [==============================] - 643s 473ms/step - loss: 0.4382 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 16/25
1360/1360 [==============================] - 646s 475ms/step - loss: 0.4338 - accurac
recall:  0.11610486891385768 - f1Score: 0.1962025316455696 - AUC: 0.5484982943295403
Epoch 17/25
1360/1360 [==============================] - 647s 476ms/step - loss: 0.4111 - accurac
recall:  0.19101123595505617 - f1Score: 0.2982456140350877 - AUC: 0.5827667644743434
Epoch 18/25
1360/1360 [==============================] - 639s 470ms/step - loss: 0.4198 - accurac
recall:  0.25842696629213485 - f1Score: 0.375 - AUC: 0.6122283451418211
Epoch 19/25
1360/1360 [==============================] - 647s 476ms/step - loss: 0.4114 - accurac
recall:  0.449438202247191 - f1Score: 0.45197740112994345 - AUC: 0.6482859801044873
Epoch 20/25
1360/1360 [==============================] - 643s 473ms/step - loss: 0.3909 - accurac
recall:  0.1797752808988764 - f1Score: 0.27507163323782235 - AUC: 0.5718409313199265
Epoch 21/25
1360/1360 [==============================] - 644s 474ms/step - loss: 0.3787 - accurac
recall:  0.00749063670411985 - f1Score: 0.014705882352941176 - AUC: 0.502152961664161
Epoch 22/25
1360/1360 [==============================] - 663s 488ms/step - loss: 0.3872 - accurac
recall:  0.0449438202247191 - f1Score: 0.08362369337979093 - AUC: 0.518225625611298
Epoch 23/25
1360/1360 [==============================] - 650s 478ms/step - loss: 0.3794 - accurac
recall:  0.4606741573033708 - f1Score: 0.5061728395061729 - AUC: 0.6793816646389466
Epoch 24/25
1360/1360 [==============================] - 655s 482ms/step - loss: 0.3641 - accurac
recall:  0.18352059925093633 - f1Score: 0.28654970760233917 - AUC: 0.5779598749970181
Epoch 25/25
1360/1360 [==============================] - 664s 488ms/step - loss: 0.3548 - accurac
```

```
#this is after changed pipeline
history3 = model3.fit(train_data, steps_per_epoch=len(train_data), epochs=1,\
                              validation_data=test_data,callbacks=callbacks, )
```

```
   2/1360 [..............................] - ETA: 55:39 - loss: 0.6746 - accuracy: 0.875
```

```
history3 = model3.fit(train_data, steps_per_epoch=len(train_data), epochs=25,\
                              validation_data=test_data,callbacks=callbacks, )
```

```
Epoch 6/25
1360/1360 [==============================] - 635s 467ms/step - loss: 0.5231 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 7/25
1360/1360 [==============================] - 633s 465ms/step - loss: 0.5227 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 8/25
1360/1360 [==============================] - 632s 464ms/step - loss: 0.5216 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 9/25
1360/1360 [==============================] - 639s 470ms/step - loss: 0.5199 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 10/25
1360/1360 [==============================] - 635s 467ms/step - loss: 0.5210 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 11/25
1360/1360 [==============================] - 644s 473ms/step - loss: 0.5161 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 12/25
1360/1360 [==============================] - 635s 467ms/step - loss: 0.5096 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 13/25
1360/1360 [==============================] - 636s 467ms/step - loss: 0.5082 - accurac
recall:  0.003745318352059925 - f1Score: 0.007434944237918215 - AUC: 0.50134187361339
Epoch 14/25
1360/1360 [==============================] - 636s 467ms/step - loss: 0.5039 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.4989384288747346
Epoch 15/25
1360/1360 [==============================] - 637s 469ms/step - loss: 0.4923 - accurac
recall:  0.0 - f1Score: 0.0 - AUC: 0.5
Epoch 16/25
1360/1360 [==============================] - 637s 468ms/step - loss: 0.4799 - accurac
recall:  0.14606741573033707 - f1Score: 0.2154696132596685 - AUC: 0.5433097163577375
Epoch 17/25
1360/1360 [==============================] - 641s 472ms/step - loss: 0.4911 - accurac
recall:  0.026217228464419477 - f1Score: 0.04778156996587031 - AUC: 0.503023688542188

Epoch 18/25
1360/1360 [==============================] - 642s 472ms/step - loss: 0.4784 - accurac
recall:  0.0599250936329588 - f1Score: 0.1111111111111111 - AUC: 0.5273086190033158
Epoch 19/25
1360/1360 [==============================] - 653s 480ms/step - loss: 0.4664 - accurac
recall:  0.026217228464419477 - f1Score: 0.04982206405693951 - AUC: 0.509393115293781
Epoch 20/25
1360/1360 [==============================] - 650s 478ms/step - loss: 0.4580 - accurac
recall:  0.27715355805243447 - f1Score: 0.35576923076923084 - AUC: 0.5987678618287651
Epoch 21/25
1360/1360 [==============================] - 654s 481ms/step - loss: 0.4521 - accurac
recall:  0.033707865168539325 - f1Score: 0.06405693950177936 - AUC: 0.514200004771106
Epoch 22/25
1360/1360 [==============================] - 650s 478ms/step - loss: 0.4495 - accurac
recall:  0.2247191011235955 - f1Score: 0.3225806451612903 - AUC: 0.5884742002433265
```

```
Epoch 23/25
1360/1360 [==============================] - 654s 481ms/step - loss: 0.4376 - accurac
recall:  0.1947565543071161 - f1Score: 0.29714285714285715 - AUC: 0.5809239247119444
Epoch 24/25
1360/1360 [==============================] - 663s 488ms/step - loss: 0.4259 - accurac
recall:  0.4794007490636704 - f1Score: 0.48669201520912553 - AUC: 0.670167465826952
Epoch 25/25
```

```
history3 = model3.fit(train_data, steps_per_epoch=len(train_data), epochs=20,\
                              validation_data=test_data,callbacks=callbacks, )
```

```
Epoch 1/20
1360/1360 [==============================] - 681s 500ms/step - loss: 0.4165 - accurac
recall:  0.38202247191011235 - f1Score: 0.4646924829157176 - AUC: 0.6538562465707675
Epoch 2/20
1360/1360 [==============================] - 686s 504ms/step - loss: 0.4053 - accurac
recall:  0.2958801498127341 - f1Score: 0.3940149625935162 - AUC: 0.6187468689615688
Epoch 3/20
1360/1360 [==============================] - 700s 515ms/step - loss: 0.4017 - accurac
recall:  0.5243445692883895 - f1Score: 0.5303030303030303 - AUC: 0.6979472315656384
Epoch 4/20
1360/1360 [==============================] - 691s 508ms/step - loss: 0.3959 - accurac
recall:  0.3408239700374532 - f1Score: 0.44067796610169496 - AUC: 0.6412187790739283
Epoch 5/20
1360/1360 [==============================] - 697s 512ms/step - loss: 0.3892 - accurac
recall:  0.4307116104868914 - f1Score: 0.5066079295154186 - AUC: 0.6771392447338915
Epoch 6/20
1360/1360 [==============================] - 695s 511ms/step - loss: 0.3911 - accurac
recall:  0.49063670411985016 - f1Score: 0.5229540918163673 - AUC: 0.6906474391087575
Epoch 7/20
1360/1360 [==============================] - 713s 524ms/step - loss: 0.3845 - accurac
recall:  0.25842696629213485 - f1Score: 0.3822714681440444 - AUC: 0.6159438440802499
Epoch 8/20
1360/1360 [==============================] - 692s 509ms/step - loss: 0.3826 - accurac
recall:  0.4157303370786517 - f1Score: 0.49443207126948785 - AUC: 0.6701793935924043
Epoch 9/20
1360/1360 [==============================] - 691s 508ms/step - loss: 0.3745 - accurac
recall:  0.3595505617977528 - f1Score: 0.4393592677345538 - AUC: 0.6404971492640569
Epoch 10/20
1360/1360 [==============================] - 687s 505ms/step - loss: 0.3644 - accurac
recall:  0.4344569288389513 - f1Score: 0.5178571428571428 - AUC: 0.6827274028483503
Epoch 11/20
1360/1360 [==============================] - 679s 499ms/step - loss: 0.3602 - accurac
recall:  0.3857677902621723 - f1Score: 0.4963855421686747 - AUC: 0.6689985448126149
Epoch 12/20
1360/1360 [==============================] - 673s 495ms/step - loss: 0.3516 - accurac
recall:  0.42696629213483145 - f1Score: 0.5241379310344828 - AUC: 0.6848207256852501
Epoch 13/20
1360/1360 [==============================] - 637s 468ms/step - loss: 0.3426 - accurac
recall:  0.5543071161048689 - f1Score: 0.5077186963979418 - AUC: 0.6879815835301415
Epoch 14/20
1360/1360 [==============================] - 642s 472ms/step - loss: 0.3465 - accurac
recall:  0.3857677902621723 - f1Score: 0.4768518518518518 - AUC: 0.659975190247859
Epoch 15/20
1360/1360 [==============================] - 666s 490ms/step - loss: 0.3211 - accurac
recall:  0.3970037453183521 - f1Score: 0.4988235294117647 - AUC: 0.6709010234022759
```

```
Epoch 16/20
1360/1360 [==============================] - 653s 480ms/step - loss: 0.3081 - accurac
recall:  0.4794007490636704 - f1Score: 0.5267489711934157 - AUC: 0.6913988883322598
Epoch 17/20
1360/1360 [==============================] - 660s 485ms/step - loss: 0.2942 - accurac
recall:  0.48314606741573035 - f1Score: 0.516 - AUC: 0.6863713351940648
Epoch 18/20
1360/1360 [==============================] - 647s 476ms/step - loss: 0.2873 - accurac
recall:  0.5056179775280899 - f1Score: 0.531496062992126 - AUC: 0.6965457191249792
Epoch 19/20
1360/1360 [==============================] - 657s 483ms/step - loss: 0.2747 - accurac
recall:  0.35580524344569286 - f1Score: 0.46798029556650245 - AUC: 0.6545480569670078
Epoch 20/20
```

```
model3.summary()
```

```
Model: "resnet34_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           multiple                  4736
_________________________________________________________________
batch_normalization_36 (Batc multiple                  128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 multiple                  0
_________________________________________________________________
sequential_7 (Sequential)    (None, 64, 64, 32)        56256
_________________________________________________________________
sequential_8 (Sequential)    (None, 32, 32, 64)        281408
_________________________________________________________________
sequential_10 (Sequential)   (None, 16, 16, 128)       1712256
_________________________________________________________________
sequential_12 (Sequential)   (None, 8, 8, 256)         3285760
_________________________________________________________________
average_pooling2d_1 (Average multiple                  0
_________________________________________________________________
flatten_1 (Flatten)          multiple                  0
_________________________________________________________________
dense_3 (Dense)              multiple                  1229100
_________________________________________________________________
dense_4 (Dense)              multiple                  15050
_________________________________________________________________
dense_5 (Dense)              multiple                  51
=================================================================
Total params: 6,584,745
Trainable params: 6,576,233
Non-trainable params: 8,512
_________________________________________________________________
```

```
model3.load_weights("best_model3.h5")
```

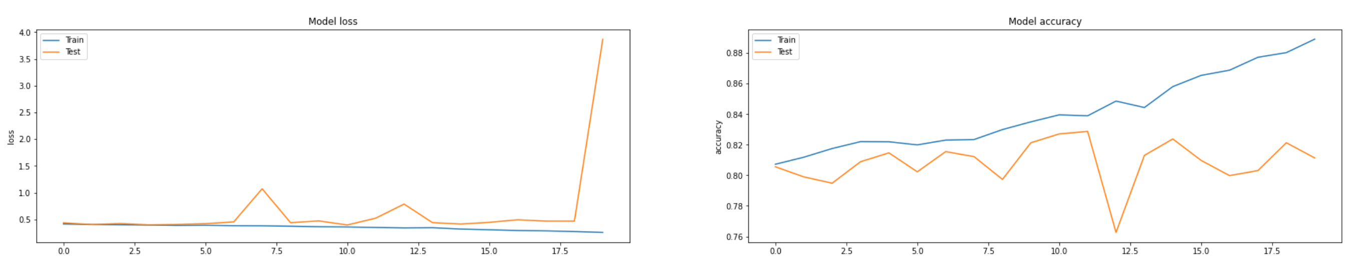
```
#Saving the model
model3.save("pneumothorax_classifier")
```

```
WARNING:absl:Found untraced functions such as conv2d_37_layer_call_and_return_conditiona
WARNING:absl:Found untraced functions such as conv2d_37_layer_call_and_return_conditiona
INFO:tensorflow:Assets written to: pneumothorax_classifier/assets
INFO:tensorflow:Assets written to: pneumothorax_classifier/assets
```

```
!zip -r /content/pneumothorax_classifier.zip /content/pneumothorax_classifier/
```

```
  adding: content/pneumothorax_classifier/ (stored 0%)
  adding: content/pneumothorax_classifier/variables/ (stored 0%)
  adding: content/pneumothorax_classifier/variables/variables.index (deflated 80%)
  adding: content/pneumothorax_classifier/variables/variables.data-00000-of-00001 (defla
  adding: content/pneumothorax_classifier/assets/ (stored 0%)
  adding: content/pneumothorax_classifier/saved_model.pb (deflated 92%)
```

```
# Plot training & validation iou_score values
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(122)
plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



-- Oversampling the images which are predicted wrongly

```
train.iloc[0]['images_paths']
```

```
'siim/dicom-images-train-jpg/1.jpg'
```

```
imagesIndexes = []
for i in tqdm(range(len(train))):
    image = tf.keras.preprocessing.image.load_img(train.iloc[i]['images_paths'])
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)
    image = tf.expand_dims(image, axis = 0)
    prediction = model3.predict(image)

    if np.squeeze(prediction , -1).round() != train.iloc[i]['has_pneumothorax']:
        if prediction <= 0.3 or prediction >= 0.7:
            imagesIndexes.append(i)
```

```
100%|██████████| 12089/12089 [25:01<00:00,  8.05it/s]
```

```
len(imagesIndexes)
```

```
910
```

```
fileNames = train['images_paths']
labels = train['has_pneumothorax']
```

```
fileNames.shape
```

```
(12089,)
```

```
#Copy pasting so that these images appear two times
wrongPredictionImages = pd.Series(train.iloc[imagesIndexes]['images_paths'])
wrongPredictionLabels = pd.Series(train.iloc[imagesIndexes]['has_pneumothorax'])
```

```
fileNames = fileNames.append(wrongPredictionImages, ignore_index=True)
labels = labels.append(wrongPredictionLabels, ignore_index= True)
```

```
fileNames.shape
```

```
(12999,)
```

```
fileNames.tail(2)
```

```
12997    siim/dicom-images-train-jpg/12053.jpg
12998    siim/dicom-images-train-jpg/12073.jpg
Name: images_paths, dtype: object
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(fileNames, labels, test_size=0.10, strati
```

```
def read_image(datapoint):
    image = tf.keras.preprocessing.image.load_img(datapoint)
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)     # contrast correction
    return image

def read_label(datapoint):
    datapoint = tf.reshape(datapoint, [1])
    return datapoint

def preprocess(image, label):
    def f(image, label):

        image = image.decode()
        image = read_image(image)

        a = np.random.uniform()
        if a>=0.80:
            b = np.random.uniform()
            if b<0.50:
                image = tf.image.flip_left_right(image)
            else:
                image = tf.image.flip_up_down(image)

        label = read_label(label)
        return image, label
    images, labels = tf.numpy_function(f, [image, label], [tf.float32, tf.int64])
    images.set_shape([img_size, img_size, 3])
    labels.set_shape([1])
    return images, labels

def tf_dataset(x, y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=15000)
    dataset = dataset.map(preprocess)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(2)
    return dataset
```

```
train_data = tf_dataset(X_train.values, y_train.values)
```

```
def read_image(datapoint):
    image = tf.keras.preprocessing.image.load_img(datapoint)
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)     # contrast correction
    return image

def read_label(datapoint):
    datapoint = tf.reshape(datapoint, [1])
    return datapoint

def preprocess(image, label):
    def f(image, label):

        image = image.decode()
        image = read_image(image)

        label = read_label(label)
        return image, label
    images, labels = tf.numpy_function(f, [image, label], [tf.float32, tf.int64])
    images.set_shape([img_size, img_size, 3])
    labels.set_shape([1])
    return images, labels

def tf_dataset(x, y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=15000)
    dataset = dataset.map(preprocess)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(2)
    return dataset
```

```
test_data = tf_dataset(X_test.values, y_test.values)
```

```
model4 = Resnet34((256, 256, 3),[3, 4, 6, 3])
```

```
callbacks = [
    tf.keras.callbacks.ModelCheckpoint('./best_model4.h5', save_weights_only=True, save_best_
                                       mode='min'),
             history_Model1
]
optimizer = tf.keras.optimizers.Adam(0.001)
```

```
model4.compile(optimizer, loss = 'binary_crossentropy' , metrics=['accuracy'])
```

```
#this is after changed pipeline
```

```
history4 = model4.fit(train_data, steps_per_epoch=len(train_data), epochs=1,\
                              validation_data=test_data,callbacks=callbacks, )
```

```
   2/1360 [..............................] - ETA: 52:28 - loss: 2.5245 - accuracy: 0.500
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-46-403a4323ca77> in <module>()
      1 #this is after changed pipeline
----> 2 history4 = model4.fit(train_data, steps_per_epoch=len(train_data), epochs=1,
validation_data=test_data,callbacks=callbacks, )

                              6 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

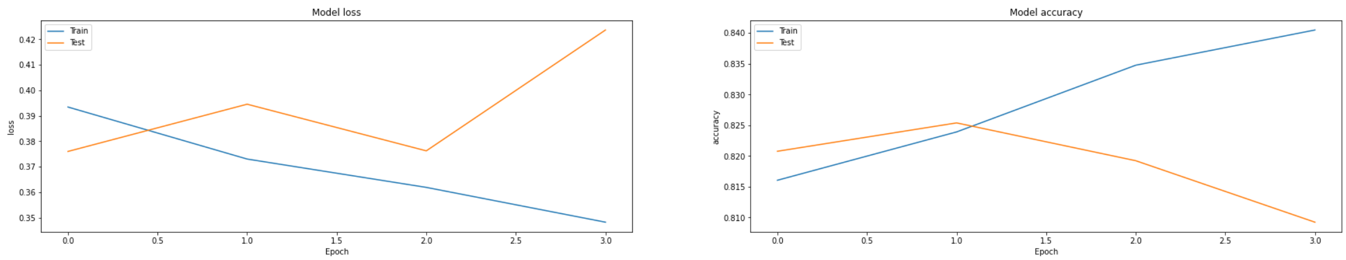
```
model4.load_weights("best_model3.h5")
```

```
history4 = model4.fit(train_data, steps_per_epoch=len(train_data), epochs=4,\
                              validation_data=test_data,callbacks=callbacks, )
```

```
Epoch 1/4
1463/1463 [==============================] - 717s 490ms/step - loss: 0.3934 - accuracy:
recall:  0.5644699140401146 - f1Score: 0.6283891547049442 - AUC: 0.7396482062314137
Epoch 2/4
1463/1463 [==============================] - 717s 490ms/step - loss: 0.3730 - accuracy:
recall:  0.6819484240687679 - f1Score: 0.6770981507823612 - AUC: 0.7799857788062031
Epoch 3/4
1463/1463 [==============================] - 725s 496ms/step - loss: 0.3619 - accuracy:
recall:  0.4899713467048711 - f1Score: 0.5927209705372616 - AUC: 0.7150172190937605
Epoch 4/4
1463/1463 [==============================] - 721s 493ms/step - loss: 0.3482 - accuracy:
recall:  0.47564469914040114 - f1Score: 0.5724137931034482 - AUC: 0.7036477964682026
```

```
# Plot training & validation iou_score values
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
```

```
# Plot training & validation loss values
plt.subplot(122)
plt.plot(history4.history['accuracy'])
plt.plot(history4.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
imagesIndexesOverSample = []
for i in tqdm(range(len(train))):
    image = tf.keras.preprocessing.image.load_img(train.iloc[i]['images_paths'])
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)
    image = tf.expand_dims(image, axis = 0)
    prediction = model4.predict(image)

    if np.squeeze(prediction , -1).round() != train.iloc[i]['has_pneumothorax']:
        if prediction <= 0.3 or prediction >= 0.7:
            imagesIndexesOverSample.append(i)
```

```
100%|██████████| 12089/12089 [25:05<00:00,  8.03it/s]
```

```
len(imagesIndexesOverSample)
```

```
484
```

```
#Saving the model
model4.save("pneumothorax_classifier_oversampling")
```

```
WARNING:absl:Found untraced functions such as conv2d_37_layer_call_and_return_conditiona
WARNING:absl:Found untraced functions such as conv2d_37_layer_call_and_return_conditiona
```

```
INFO:tensorflow:Assets written to: pneumothorax_classifier_oversampling/assets
INFO:tensorflow:Assets written to: pneumothorax_classifier_oversampling/assets
```

```
!zip -r /content/pneumothorax_classifier_oversampling.zip /content/pneumothorax_classifier_ov
```

```
  adding: content/pneumothorax_classifier_oversampling/ (stored 0%)
  adding: content/pneumothorax_classifier_oversampling/variables/ (stored 0%)
  adding: content/pneumothorax_classifier_oversampling/variables/variables.index (deflat
  adding: content/pneumothorax_classifier_oversampling/variables/variables.data-00000-of
  adding: content/pneumothorax_classifier_oversampling/assets/ (stored 0%)
  adding: content/pneumothorax_classifier_oversampling/saved_model.pb (deflated 91%)
```

Training on highly missclassified points reduced our highly missclassified points to half.

Comparison of two models

```
from random import sample
listOfIndices = sample(imagesIndexes, 2)
listOfIndices
```

```
[7566, 9532]
```

```
for index in listOfIndices:
    image = tf.keras.preprocessing.image.load_img(train.iloc[index]['images_paths'])
    image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
    image = tf.image.resize(image, [img_size, img_size])
    image = image / 255.0
    image = exposure.equalize_adapthist(image)
    image = tf.expand_dims(image, axis = 0)
    #Old model
    prediction = model3.predict(image)
    print("Prediction using old model - (before oversampling) Actual Value :" , train.iloc[in
    #New model
    prediction = model4.predict(image)
    print("Prediction using new model - (after oversampling)  Actual Value :" , train.iloc[in
    print("\n")
```

```
Prediction using old model - (before oversampling) Actual Value : 1  predicted value: [
Prediction using new model - (after oversampling)  Actual Value : 1  predicted value: [


Prediction using old model - (before oversampling) Actual Value : 1  predicted value: [
Prediction using new model - (after oversampling)  Actual Value : 1  predicted value: [
```

- Model is performing better after oversampling

Lets try changing the threshold

```
model4.load_weights("best_model4.h5")
```

```
def roundingOff(value, threshold):
    return 1 if value >=threshold else 0

def findingBestThreshold(thresholdList):
    truePositives = np.zeros(len(thresholdList))
    falseNegatives = np.zeros(len(thresholdList))
    for i in tqdm(range(len(train))):
        image = tf.keras.preprocessing.image.load_img(train.iloc[i]['images_paths'])
        image = tf.keras.preprocessing.image.img_to_array(image, dtype='float32')
        image = tf.image.resize(image, [img_size, img_size])
        image = image / 255.0
        image = exposure.equalize_adapthist(image)
        image = tf.expand_dims(image, axis = 0)
        prediction = model4.predict(image)

        y_true = train.iloc[i]['has_pneumothorax']
        for thres in range(len(thresholdList)):
            y_pred = roundingOff(prediction, thresholdList[thres])

            if y_true == 1 and y_pred == 1:
                truePositives[thres] = truePositives[thres] + 1
            elif y_true == 1 and y_pred == 0:
                falseNegatives[thres] = falseNegatives[thres] + 1
    print()
    for thres in range(len(thresholdList)):
        print("Recall for threshold ", thresholdList[thres], "is: ", (truePositives[thres])/(
```

```
thresholdList = [0.4, 0.45, 0.5, 0.6, 0.75]
findingBestThreshold(thresholdList)
```

```
100%|██████████| 12089/12089 [31:13<00:00,  6.45it/s]
Recall for threshold  0.4 is:  0.8130385912326714
Recall for threshold  0.45 is:  0.7853128512551517
Recall for threshold  0.5 is:  0.735481453727988
Recall for threshold  0.6 is:  0.6403147246159611
Recall for threshold  0.75 is:  0.4166354439865118
```

```
thresholdList = [0.38, 0.39, 0.4, 0.41, 0.42]
findingBestThreshold(thresholdList)
```

```
100%|██████████| 12089/12089 [31:02<00:00,  6.49it/s]
```

```
Recall for threshold  0.38 is:  0.8291494941925814
Recall for threshold  0.39 is:  0.8227800674409891
Recall for threshold  0.4 is:  0.8130385912326714
Recall for threshold  0.41 is:  0.8077931809666542
Recall for threshold  0.42 is:  0.8029224428624954
```

- 0.38 is the best threshold.
- Further increasing the recall is resulting in decrease in precision so we are stopping at this point.