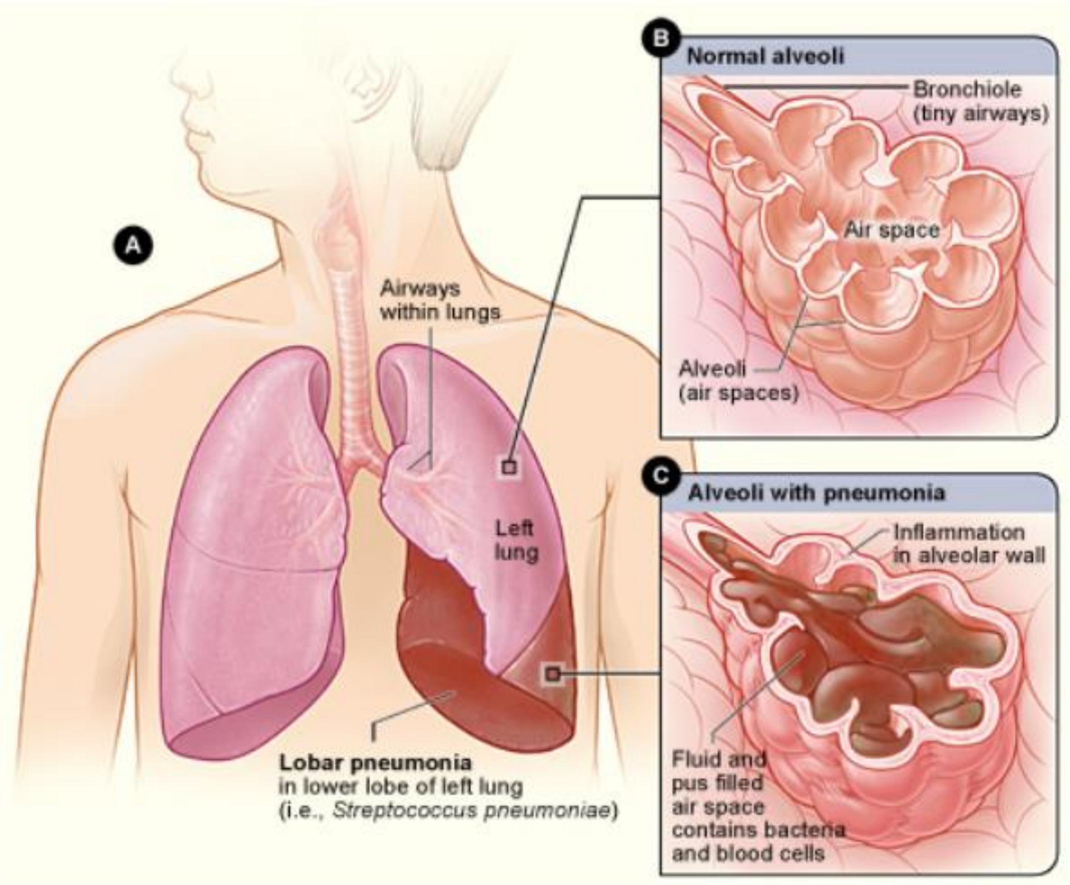
FINAL REPORT – PNEUMONIA DETECTION

WHAT IS PNUEMONIA?

Pneumonia is an infection in one or both lungs. The infection causes inflammation in the air sacs in your lungs, which are called **alveoli**. The alveoli fill with fluid or pus, making it difficult to breathe.


A
Airways within lungs
Left lung
Lobar pneumonia
in lower lobe of left lung
(i.e., *Streptococcus pneumoniae*)
B
Normal alveoli
Bronchiole (tiny airways)
Air space
Alveoli (air spaces)
C
Alveoli with pneumonia
Inflammation in alveolar wall
Fluid and pus filled air space contains bacteria and blood cells

Summary of problem statement, data and findings.

In this capstone project, we build models to detect a visual signal for pneumonia in medical images. The algorithm needs to automatically locate lung opacities on chest radiographs.

Here's the backstory and why solving this problem matters.

Pneumonia accounts for over 15% of all deaths of children under 5 years old internationally. In 2015, 920,000 children under the age of 5 died from the disease. In the United States, pneumonia accounts for over 500,000 visits to emergency departments and over 50,000 deaths in 2015, keeping the ailment on the list of top 10 causes of death in the country.

Data:

- Training Images – DICOM format
- Testing Images – DICOM format
- Detailed_Class_info – '.csv' file
- Train_Labels - '.csv' file

Our training and testing images are in DICOM (Digital Imaging and Communications in Medicine) format and are represented as ".dcm.".

More on DICOM format -

A DICOM file consists of a header and image data sets packed into a single file. The information within the header is organized as a constant and standardized series of tags. By extracting data from these tags one can access important information regarding the patient demographics, study parameters, etc. In the interest of patient confidentiality, all information that can be used to identify the patient should be removed before DICOM images are transmitted over a network for educational or other purposes.

Training data consists of 26,000+ images.

Test images consists of 3000+ images.

Detailed_Class_info – 2 columns (patientId, class)

Train_Labels - 6 columns (patientId, x, y, width, height, target)

The class column contains three categories - No Lung Opacity/Not Normal, Lung Opacity and Normal. Even though it has three categories, we will approach the problem as a binary classification problem. Because our objective is to classify if there is or not the presence of lung opacity (Pneumonia).

x, y, width and height are the bounding box's for the presence of lung opacity. If there are no values for these attributes it means there is no lung opacity for the patientId.

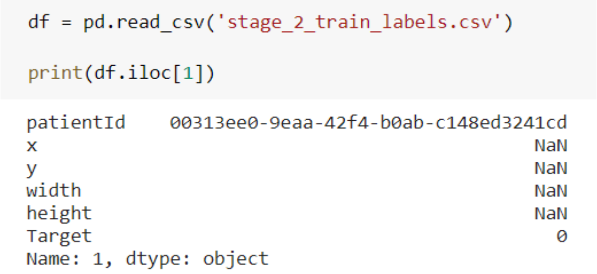
The two ".csv's" can be combined using a common attribute – patientId.

Overview of the final process.

Pre-Processing:

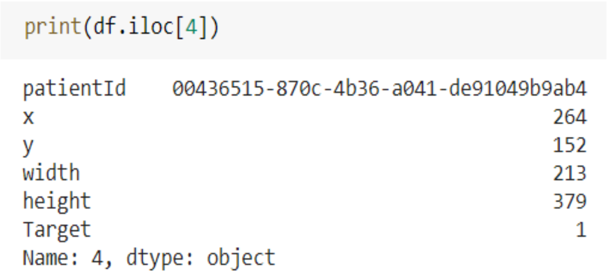
- Firstly, we analyse the individual '.csv's'.
- Combine the two csv's.
- Extract info from DICOM metadata.
- Merge with the combined csv's.
- Finally, visualize images and differences between classes via x-ray image.

Train_Labels as we see below, has information regarding bounding box's.
On left – No Lung Opactiy, and On Right – Patient with Lung Opacity



```
df = pd.read_csv('stage_2_train_labels.csv')

print(df.iloc[1])
```

```
patientId    00313ee0-9eaa-42f4-b0ab-c148ed3241cd
x                                             NaN
y                                             NaN
width                                         NaN
height                                        NaN
Target                                          0
Name: 1, dtype: object
```



```
print(df.iloc[4])
```

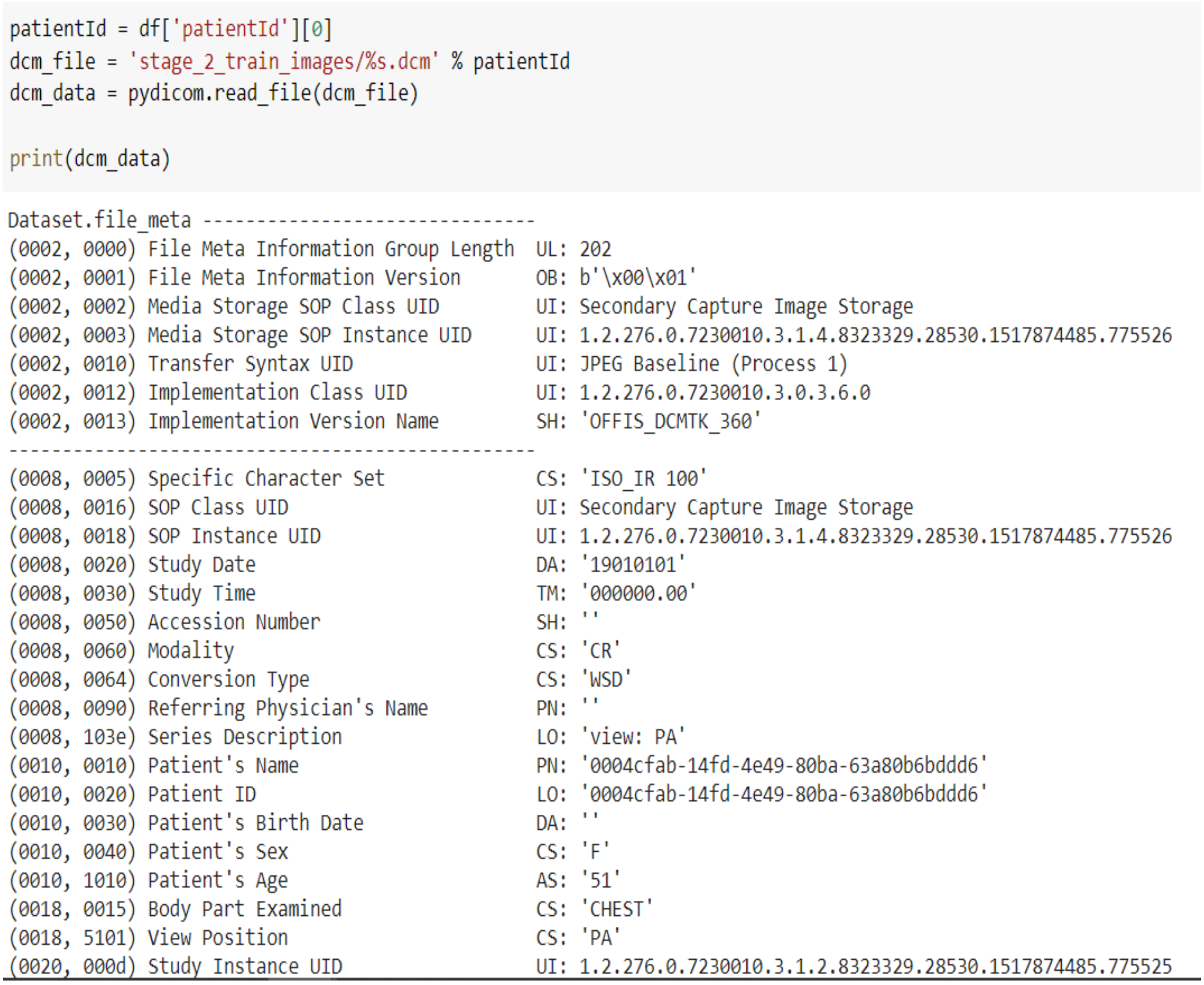
```
patientId    00436515-870c-4b36-a041-de91049b9ab4
x                                             264
y                                             152
width                                         213
height                                        379
Target                                          1
Name: 4, dtype: object
```

Also below is sample for the DICOM metadata on the first training sample:

There is a lot more information regarding the patient in the form of meta info, which can also be extracted for each patient using their id and be utilized for analysis. And we do that as well.

But not all of info on the screenshot will be utilized. *Sex*, *Age*, *Modality*, *View Position* are some of the attributes we have extracted to analyse.

DICOM:

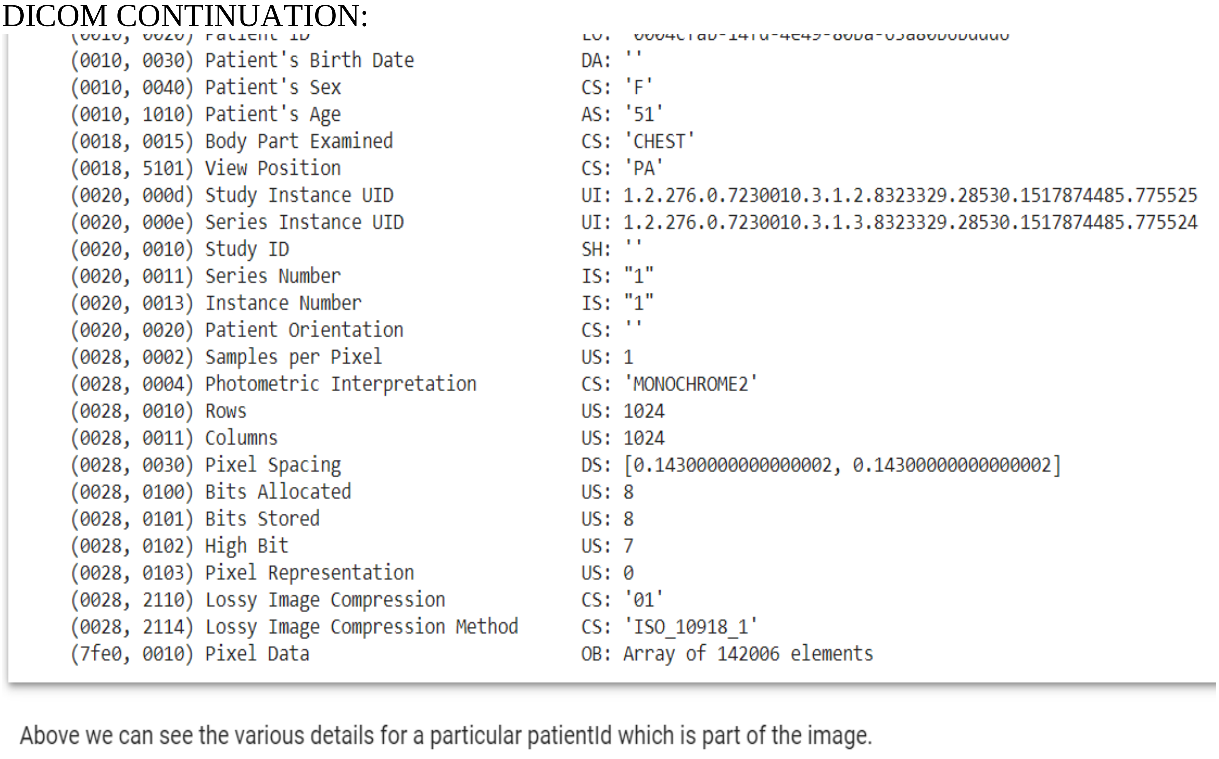


```
patientId = df['patientId'][0]
dcm_file = 'stage_2_train_images/%s.dcm' % patientId
dcm_data = pydicom.read_file(dcm_file)

print(dcm_data)
```

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 202
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
```

Final half of metadata is below. There are some valuable information present in the form of Pixel Spacing, Body Part Examined (here only single value present - Chest) etc. Pixel spacing is possible attribute which requires a thorough analysis which we have not done and is a candidate for improvement in the future.

DICOM CONTINUATION:

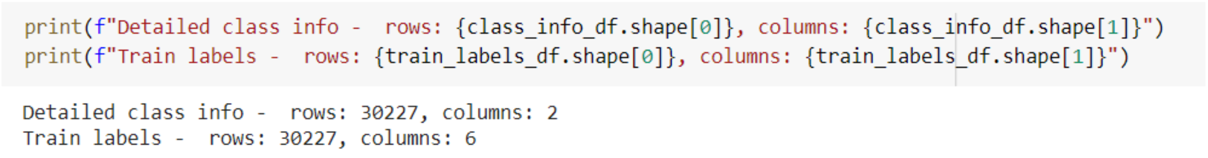


```
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: "1"
(0020, 0013) Instance Number                     IS: "1"
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.14300000000000002, 0.14300000000000002]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 142006 elements
```

Above we can see the various details for a particular patientId which is part of the image.

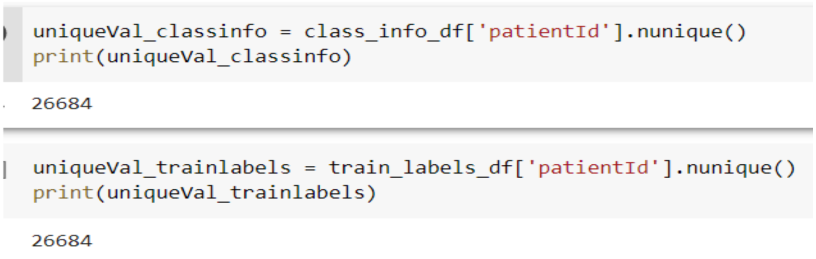
Below are two screenshot that comes of interest with respect to the counts of records and unique counts of the same for both csv's.

We noticed that unique values are short by 3000+ from the total count. From here we can inference that there is a possibility that there are multiple lung opacity location for a given patient id.



```
print(f"Detailed class info -  rows: {class_info_df.shape[0]}, columns: {class_info_df.shape[1]}")
print(f"Train labels -  rows: {train_labels_df.shape[0]}, columns: {train_labels_df.shape[1]}")
```

```
Detailed class info -  rows: 30227, columns: 2
Train labels -  rows: 30227, columns: 6
```



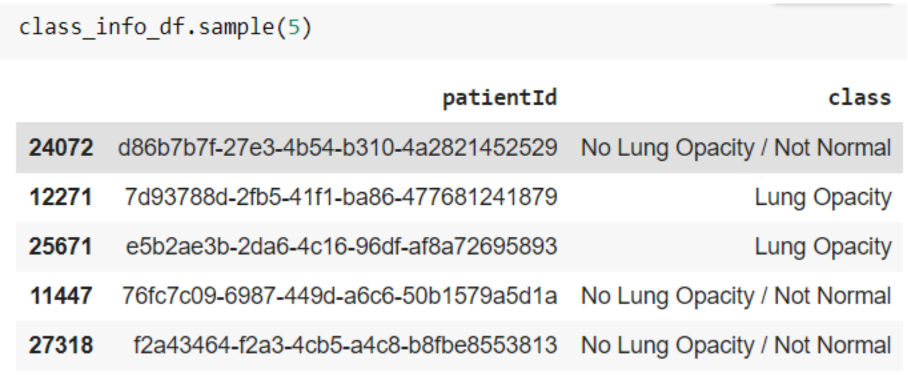
```
uniqueVal_classinfo = class_info_df['patientId'].nunique()
print(uniqueVal_classinfo)
```

```
26684
```

```
uniqueVal_trainlabels = train_labels_df['patientId'].nunique()
print(uniqueVal_trainlabels)
```

```
26684
```

Our second csv – It is important to notice that there are three unique values for 'class' – No Lung Opacity/Not Normal, Normal and Lung Opacity, we would be having binary classification not multi classification model. Cause we just have to give verdict as Yes or No for presence of lung opacity.



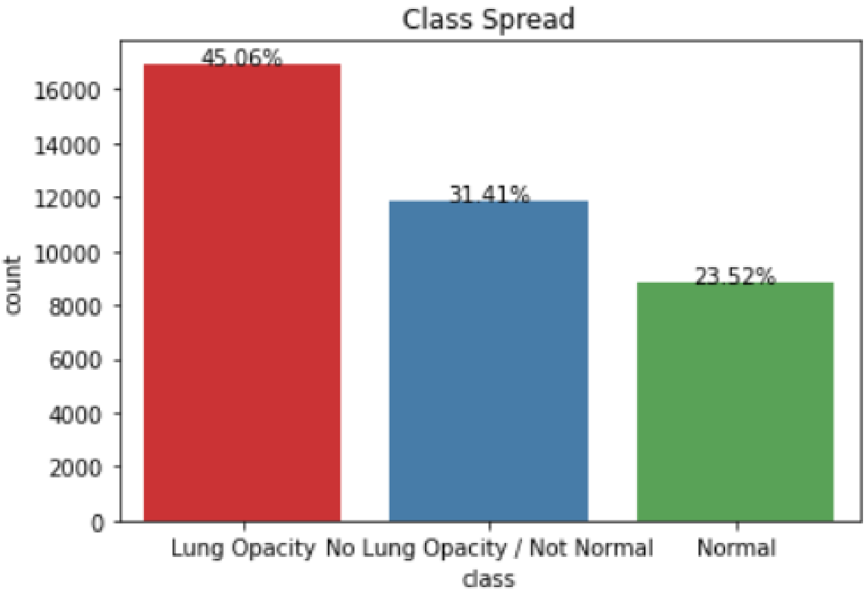
```
class_info_df.sample(5)
```

	patientId	class
24072	d86b7b7f-27e3-4b54-b310-4a2821452529	No Lung Opacity / Not Normal
12271	7d93788d-2fb5-41f1-ba86-477681241879	Lung Opacity
25671	e5b2ae3b-2da6-4c16-96df-af8a72695893	Lung Opacity
11447	76fc7c09-6987-449d-a6c6-50b1579a5d1a	No Lung Opacity / Not Normal
27318	f2a43464-f2a3-4cb5-a4c8-b8fbe8553813	No Lung Opacity / Not Normal

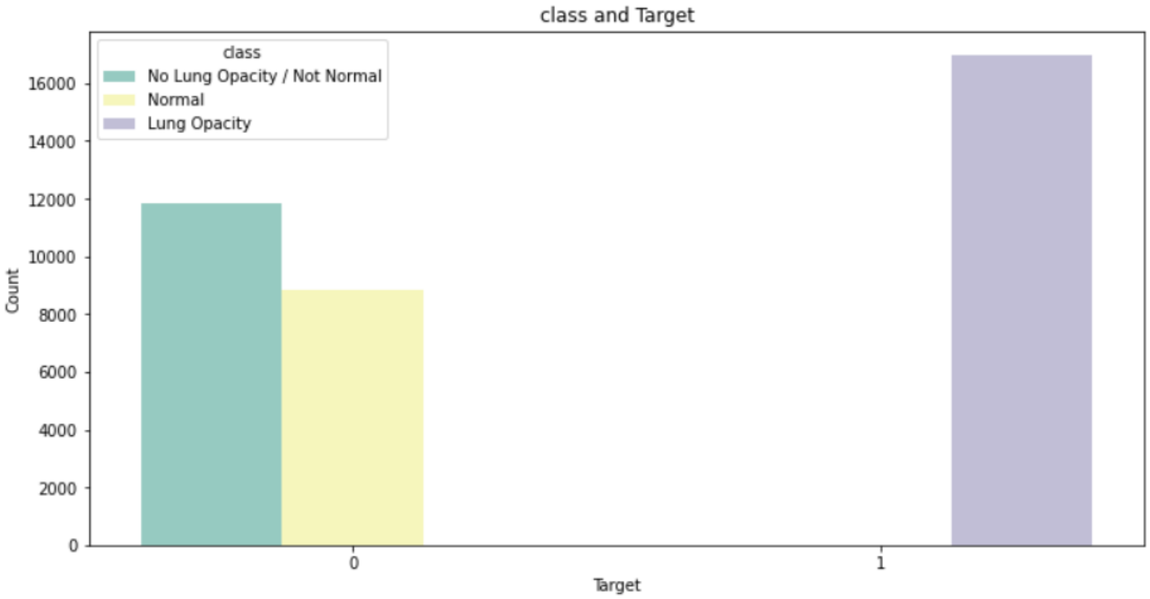
More Findings with Visualization

Some of the evidences from the dataset are shown below helping us to find the various relationships from the extracted attributes.

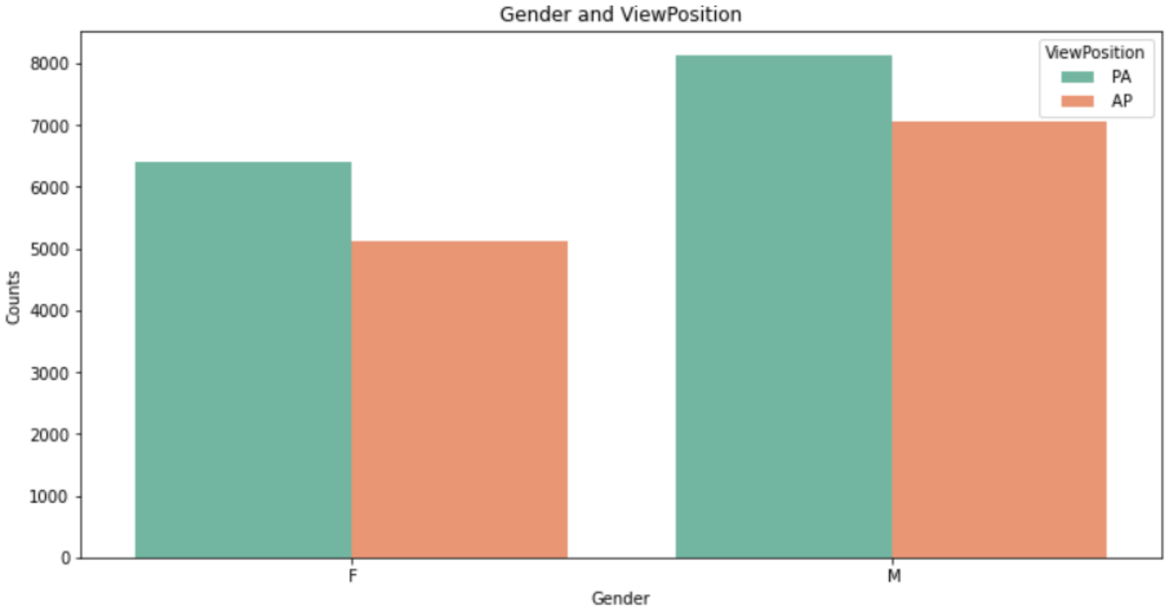
Class spread across our data set is balanced. Lung Opacity being the majority – 16000+



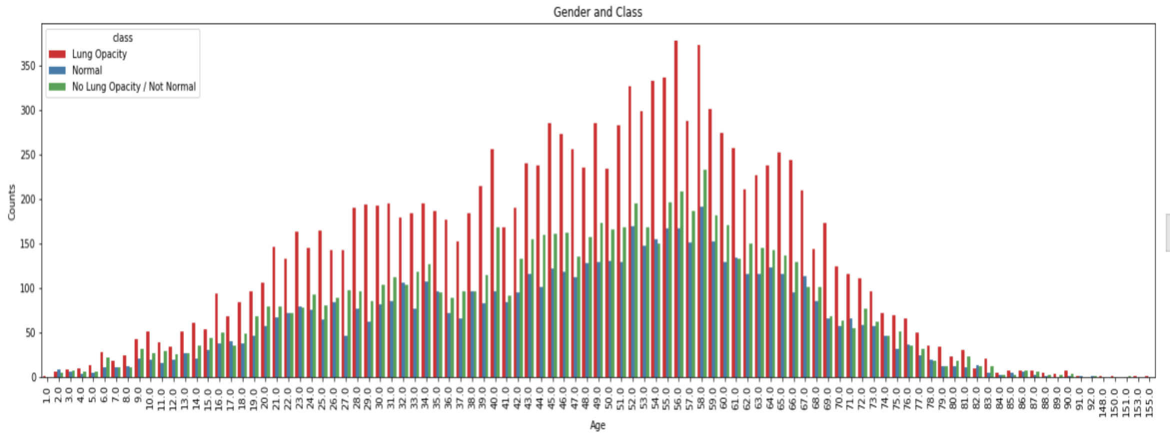
The final ‘target’ attribute which helps us making this problem a binary classification based is also well spread. There is no unbalances. Remember, a balanced dataset would help in making this best of prediction. Target ‘0’ contains two classes and ‘1’ has one – Lung Opacity.



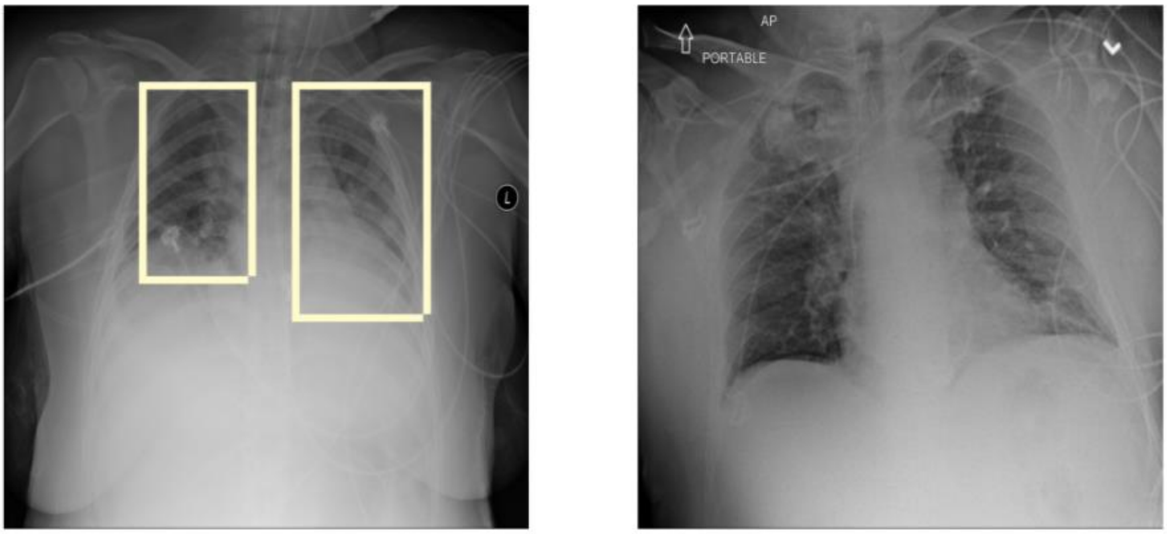
Extracted metadata attribute is compared between each other. Namely, ViewPosition and Gender. ViewPosition consists of two values – PA(Posterior Anterior) and AP(Anterior Posterior). From the below we can see Male are majority but not by much. And ViewPosition spread is very well balanced.



Below, we see the spread of age across the class attribute. We can notice ages at extremes are causing outliers. Age of 120+ and age below 3 can be considered as outliers. But we will not be correcting these outliers considering it doesn’t affect the prediction by much, but it can be considered for future enhancements.



Below X-rays are examples: On Left – Lung Opacity, On Right – Normal



Algorithms

We have tried building models using RESNET and UNET.

RESNET model we decided to be our base model for evaluation.

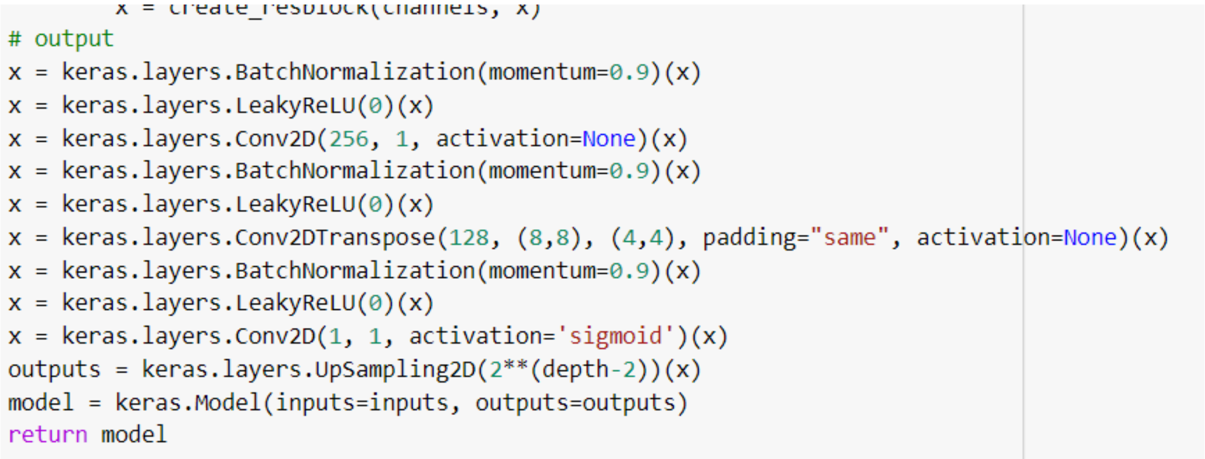
- Firstly a convolutional neural network is used to segment the image, using the bounding boxes directly as a mask.
- Secondly connected components is used to separate multiple areas of predicted pneumonia.
- Finally a bounding box is simply drawn around every connected component.

Some of the parameters that we have used on RESNET base model:

Image Size – 320

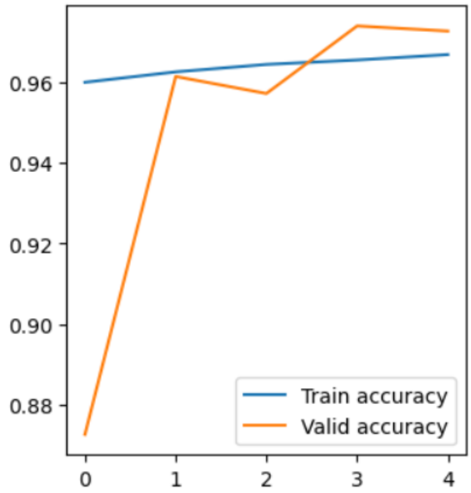
Batch Size – 16

Below is a sample from RESNET – Layers.

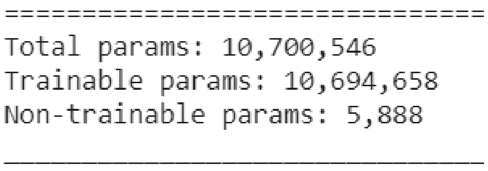


```
        x = create_resblock(channels, x)
# output
x = keras.layers.BatchNormalization(momentum=0.9)(x)
x = keras.layers.LeakyReLU(0)(x)
x = keras.layers.Conv2D(256, 1, activation=None)(x)
x = keras.layers.BatchNormalization(momentum=0.9)(x)
x = keras.layers.LeakyReLU(0)(x)
x = keras.layers.Conv2DTranspose(128, (8,8), (4,4), padding="same", activation=None)(x)
x = keras.layers.BatchNormalization(momentum=0.9)(x)
x = keras.layers.LeakyReLU(0)(x)
x = keras.layers.Conv2D(1, 1, activation='sigmoid')(x)
outputs = keras.layers.UpSampling2D(2**(depth-2))(x)
model = keras.Model(inputs=inputs, outputs=outputs)
return model
```

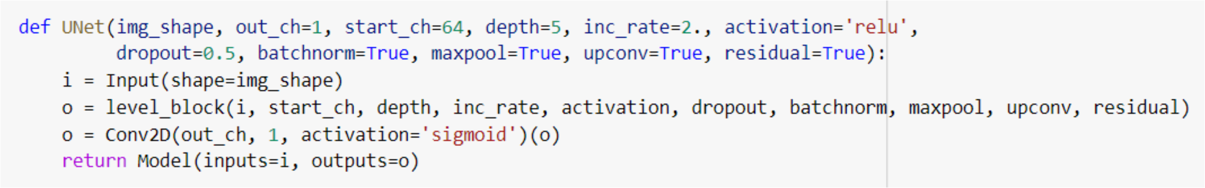
We trained RESNET for 5 epochs and achieved 97.37% validation accuracy.



For our second Model we chose UNET. We decided on UNET cause of it performance and provided great results.



```
================================
Total params: 10,700,546
Trainable params: 10,694,658
Non-trainable params: 5,888
________________________________
```



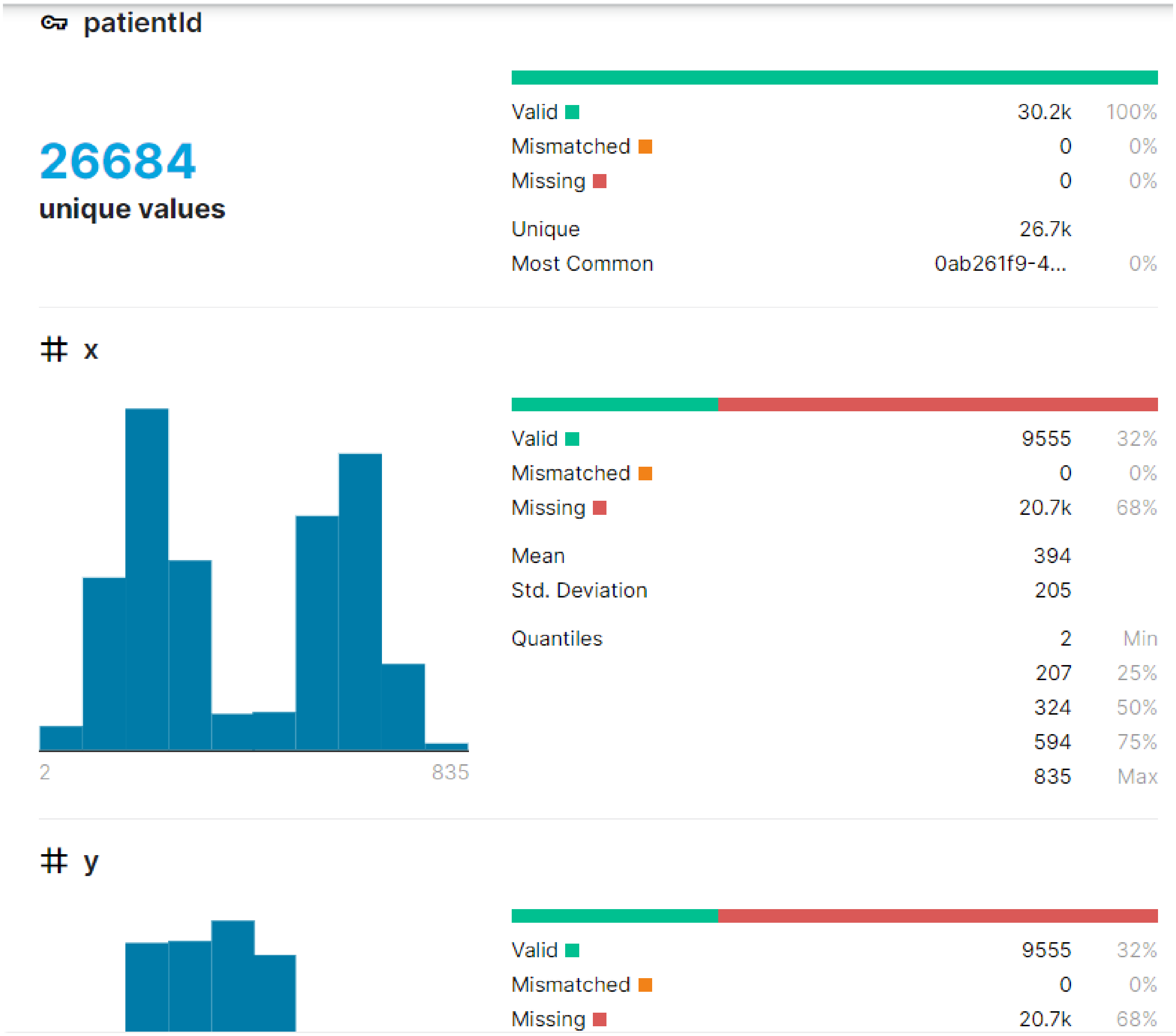
```
def UNet(img_shape, out_ch=1, start_ch=64, depth=5, inc_rate=2., activation='relu',
         dropout=0.5, batchnorm=True, maxpool=True, upconv=True, residual=True):
    i = Input(shape=img_shape)
    o = level_block(i, start_ch, depth, inc_rate, activation, dropout, batchnorm, maxpool, upconv, residual)
    o = Conv2D(out_ch, 1, activation='sigmoid')(o)
    return Model(inputs=i, outputs=o)
```

Step-by-step walk through the solution

At First, we combine csv’s based on patientid. Also verify that combined data is consistent and reliable for further use.

We begin analysis of each attribute by its count, unique values, frequency etc.(explained above under - OverView)

Checked for nulls during pre-load from Kaggle – there are no presence of missing values.



patientId

26684 unique values

Valid 🟩	30.2k	100%
Mismatched 🟧	0	0%
Missing 🟥	0	0%
Unique	26.7k	
Most Common	0ab261f9-4...	0%

x

Valid 🟩	9555	32%
Mismatched 🟧	0	0%
Missing 🟥	20.7k	68%
Mean	394	
Std. Deviation	205	
Quantiles	2	Min
	207	25%
	324	50%
	594	75%
	835	Max

y

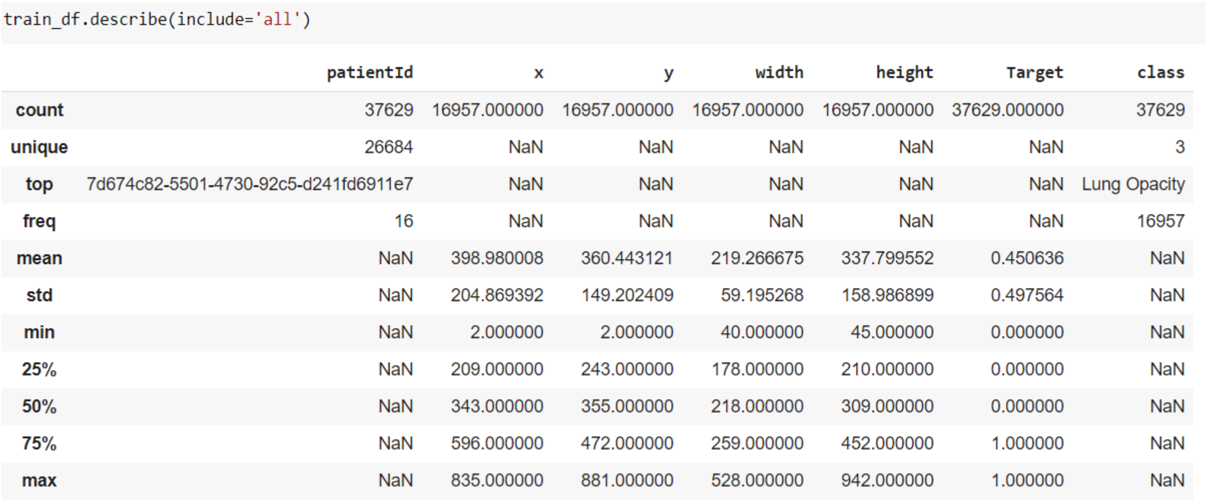
Valid 🟩	9555	32%
Mismatched 🟧	0	0%
Missing 🟥	20.7k	68%

The info can be used to understand the attributes of class_info file. There does not seem to be any missing values.

x,y,height and width is expected to have null values, cause the presence of values points the presence of opacity – pneumonia.

Combined csv’s

Earlier analysis of the data is further verified from the below result:



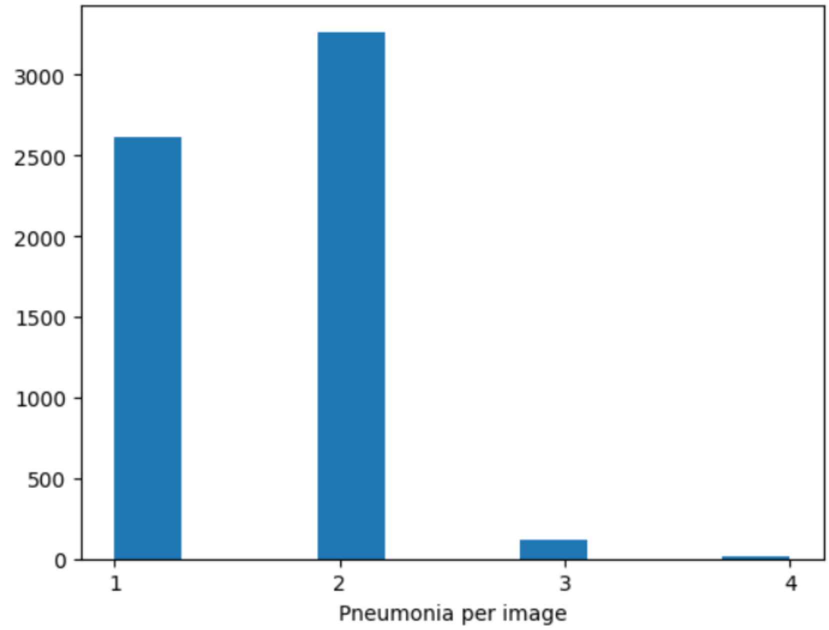
```
train_df.describe(include='all')
```

	patientId	x	y	width	height	Target	class
count	37629	16957.000000	16957.000000	16957.000000	16957.000000	37629.000000	37629
unique	26684	NaN	NaN	NaN	NaN	NaN	3
top	7d674c82-5501-4730-92c5-d241fd6911e7	NaN	NaN	NaN	NaN	NaN	Lung Opacity
freq	16	NaN	NaN	NaN	NaN	NaN	16957
mean	NaN	398.980008	360.443121	219.266675	337.799552	0.450636	NaN
std	NaN	204.869392	149.202409	59.195268	158.986899	0.497564	NaN
min	NaN	2.000000	2.000000	40.000000	45.000000	0.000000	NaN
25%	NaN	209.000000	243.000000	178.000000	210.000000	0.000000	NaN
50%	NaN	343.000000	355.000000	218.000000	309.000000	0.000000	NaN
75%	NaN	596.000000	472.000000	259.000000	452.000000	1.000000	NaN
max	NaN	835.000000	881.000000	528.000000	942.000000	1.000000	NaN

We had found that total and unique counts of the data set where different cause of presence of duplicate patientid. But this does not mean they are real duplicate tuples. X,y,width and height would have different values pointing to the fact that a patient may have more than one lung opacity.

Below analysis cements our understanding.

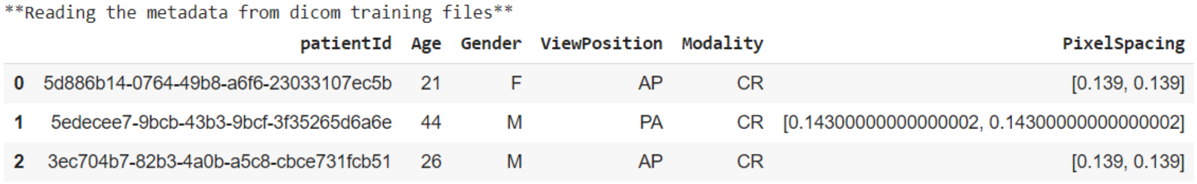
```
Total train images: 26684
Images with pneumonia: 6012
```



Dicom Metadata

We now start to discuss on DICOM metadata:

Note- Not all attributes of the metadata is useful in understanding the problem in hand.



Reading the metadata from dicom training files

	patientId	Age	Gender	ViewPosition	Modality	PixelSpacing
0	5d886b14-0764-49b8-a6f6-23033107ec5b	21	F	AP	CR	[0.139, 0.139]
1	5edecee7-9bcb-43b3-9bcf-3f35265d6a6e	44	M	PA	CR	[0.14300000000000002, 0.14300000000000002]
2	3ec704b7-82b3-4a0b-a5c8-cbce731fcb51	26	M	AP	CR	[0.139, 0.139]

Summary of metadata collected:

Gender – F,M

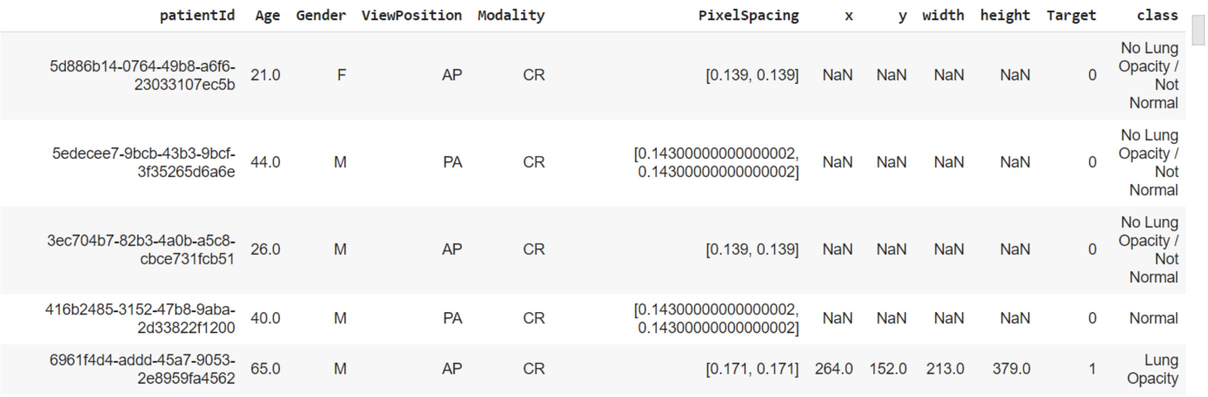
Age – Spread from 0 to 148 (After 98, next value is at above 140. Age above 140 can be considered as outliers)

ViewPosition – AP,PA (AP – Anterior/Posterior, PA – Posterior/Anterior)

Modailty – CR (Chest)

PixelSpacing – This attribute is not analysed in this kernel well. An understanding for future need.

Once the above two info is extracted we combined them. Which would help understanding the relations between age and target, which was not previously possible.



patientId	Age	Gender	ViewPosition	Modality	PixelSpacing	x	y	width	height	Target	class
5d886b14-0764-49b8-a6f6-23033107ec5b	21.0	F	AP	CR	[0.139, 0.139]	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
5edecee7-9bcb-43b3-9bcf-3f35265d6a6e	44.0	M	PA	CR	[0.14300000000000002, 0.14300000000000002]	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
3ec704b7-82b3-4a0b-a5c8-cbce731fcb51	26.0	M	AP	CR	[0.139, 0.139]	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
416b2485-3152-47b8-9aba-2d33822f1200	40.0	M	PA	CR	[0.14300000000000002, 0.14300000000000002]	NaN	NaN	NaN	NaN	0	Normal
6961f4d4-addd-45a7-9053-2e8959fa4562	65.0	M	AP	CR	[0.171, 0.171]	264.0	152.0	213.0	379.0	1	Lung Opacity

Image Analysis

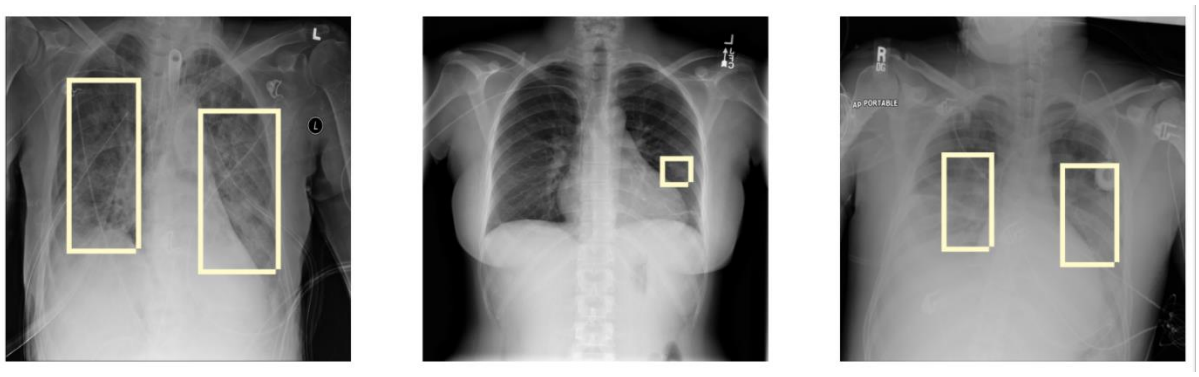
We now start analysis of DICOM images. We divided analysis as per class values.

No Lung Opacity/Not Normal (points to some other issue than pnuemonia),

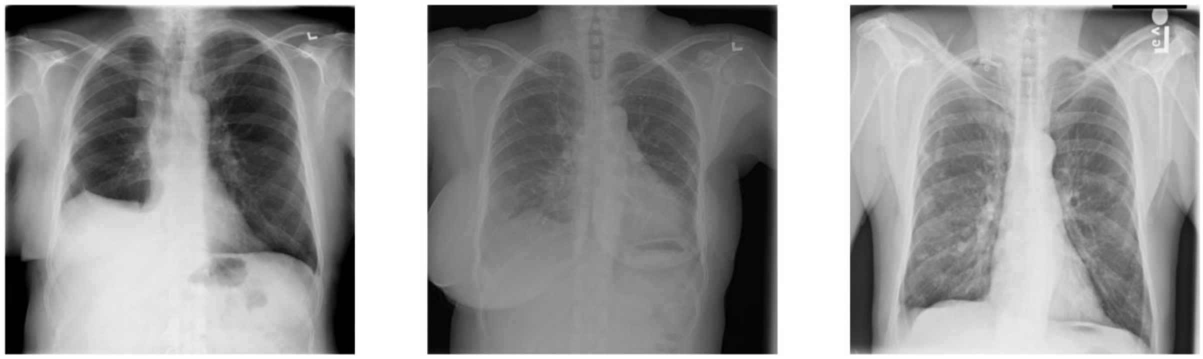
Normal (No Pnuemonia),

Lung Opacity (Pnuemonia)

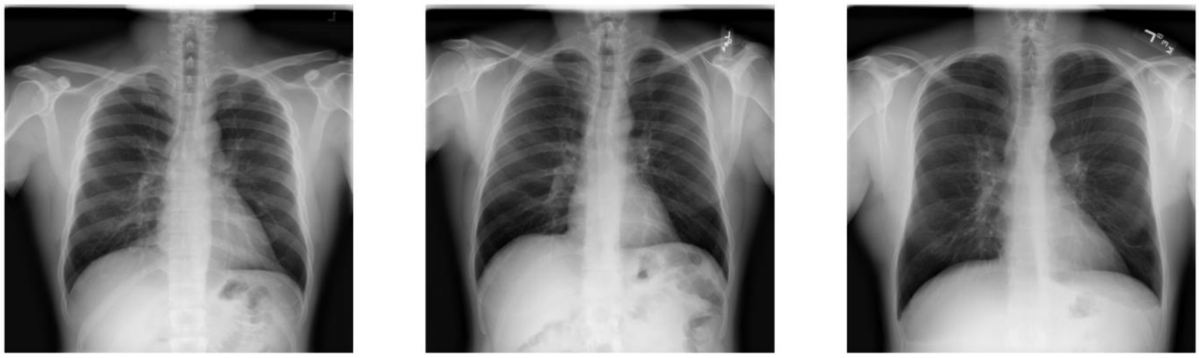
Lung Opacity



No Lung Opacity/Not Normal



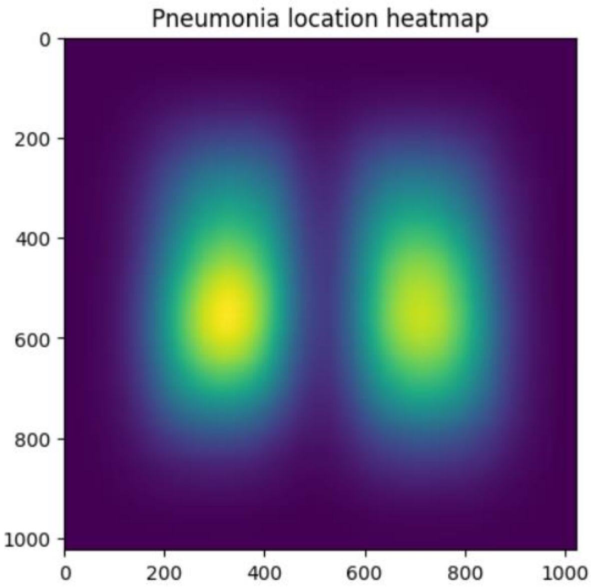
Normal:



Next, we save Pnuemonia location in a dictionary.
A list of pneumonia locations per filename is used to create heat map.

For heat map we divide training images into below as training and validation.

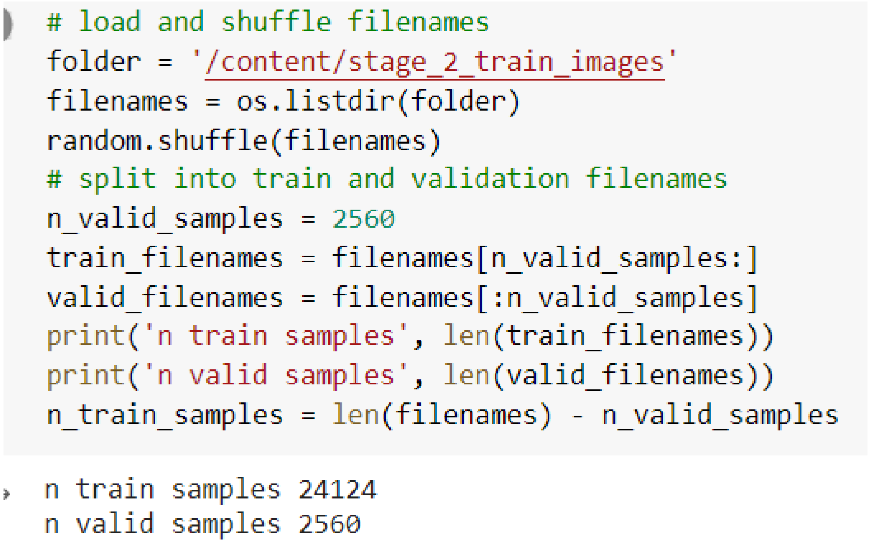
no of train samples 21684
no of valid samples 5000



Modelling

We are firstly using a basic RESNET model for evaluation purpose. We implement one basic model and analyse its performance. Later for 2nd Milestone, we try to build models with lesser layers of the same, UNET model and also another model just to check on performances.

At first, we are using below mentioned number of images for training and validation from the training set of images.



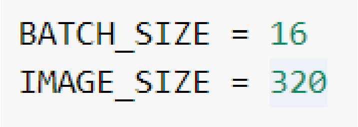
```
# load and shuffle filenames
folder = '/content/stage_2_train_images'
filenames = os.listdir(folder)
random.shuffle(filenames)
# split into train and validation filenames
n_valid_samples = 2560
train_filenames = filenames[n_valid_samples:]
valid_filenames = filenames[:n_valid_samples]
print('n train samples', len(train_filenames))
print('n valid samples', len(valid_filenames))
n_train_samples = len(filenames) - n_valid_samples
```

```
n train samples 24124
n valid samples 2560
```

The dataset is too large to fit into memory, so we create a generator that loads data on the fly.
The generator takes in some filenames, batch_size and other parameters.
The generator outputs a random batch of numpy images and numpy masks.

As part of the custom generator, we have build ‘_init_’, ‘_load_’, ‘_getitem_’, ‘on_epoch_end’ and ‘_len_’ methods.

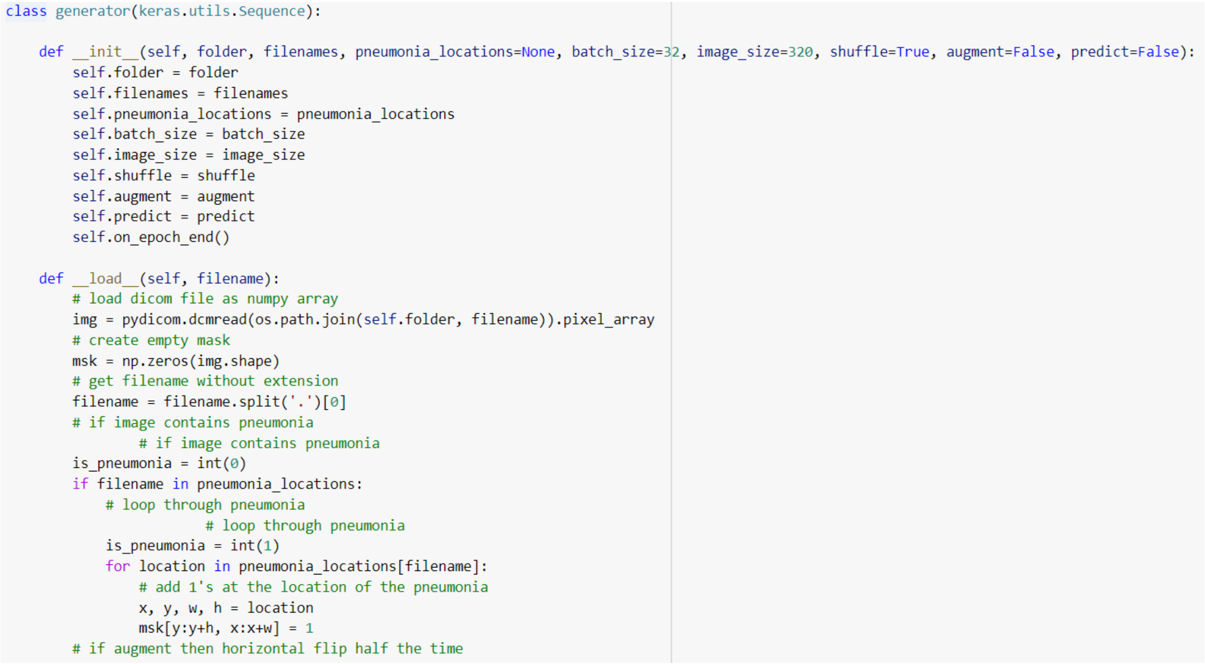
Batch Size – 16 and Image Size – 320 are the parameters used.



```
BATCH_SIZE = 16
IMAGE_SIZE = 320
```

Model 1 - RESNET

Below is the screenshot of the generator that is used.



```
class generator(keras.utils.Sequence):

    def __init__(self, folder, filenames, pneumonia_locations=None, batch_size=32, image_size=320, 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(os.path.join(self.folder, filename)).pixel_array
        # create empty mask
        msk = np.zeros(img.shape)
        # get filename without extension
        filename = filename.split('.')[0]
        # if image contains pneumonia
                # if image contains pneumonia
        is_pneumonia = int(0)
        if filename in pneumonia_locations:
            # loop through pneumonia
                        # loop through pneumonia
            is_pneumonia = int(1)
            for location in pneumonia_locations[filename]:
                # add 1's at the location of the pneumonia
                x, y, w, h = location
                msk[y:y+h, x:x+w] = 1
        # if augment then horizontal flip half the time
```

- The network consists of a number of residual blocks with convolutions and down sampling blocks with max pooling.
- At the end of the network a upsampling layer is used to convert the output to the same shape as the input.
- Also involves, transpose convolutions to allow greater flexibility

Custom Methods used are:

create_downsample (channels, inputs)

create_resblock (channels, inputs)

create_network (input_size, channels, n_blocks=2, depth=4)

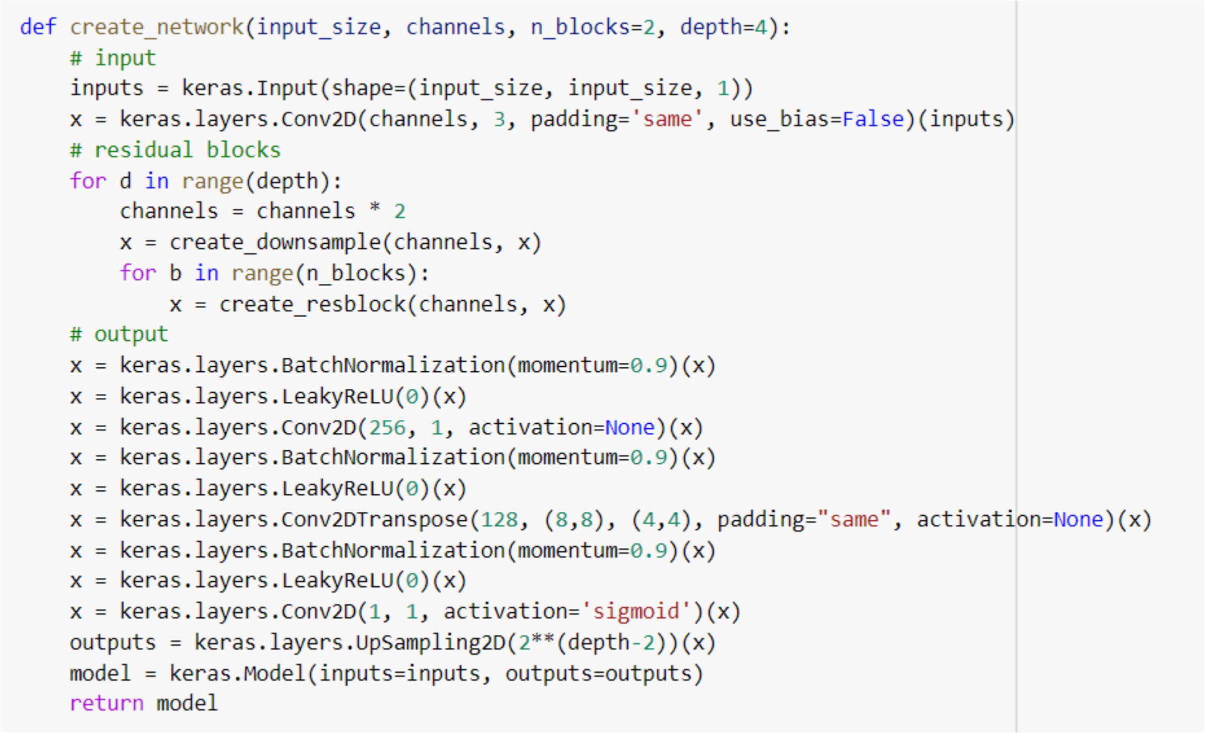
iou_loss (y_true, y_pred)

iou_bce_loss (y_true, y_pred)

mean_iou (y_true, y_pred)

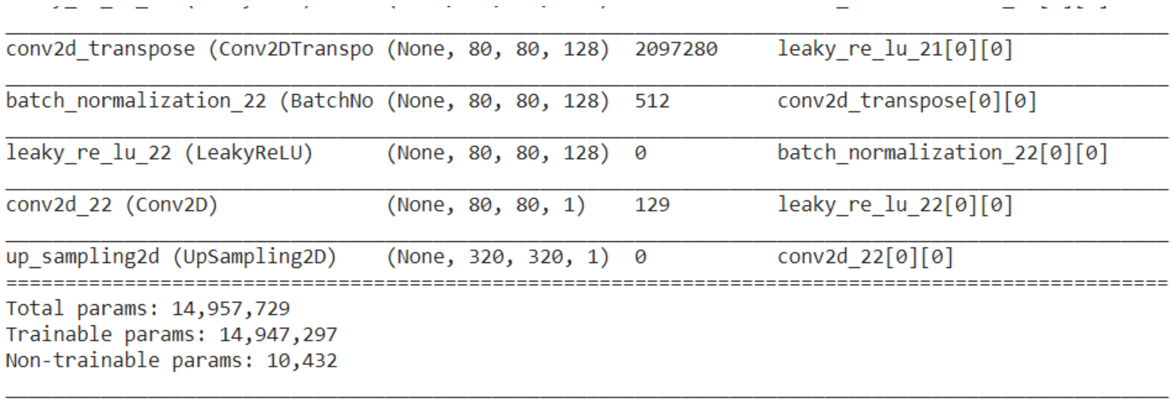
cosine_annealing (x)

Below is the layers of our first RESNET model.



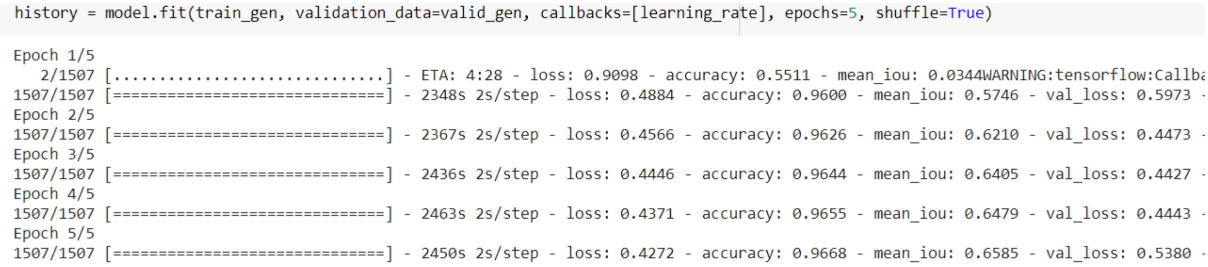
```
def create_network(input_size, channels, n_blocks=2, depth=4):
    # input
    inputs = keras.Input(shape=(input_size, input_size, 1))
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(inputs)
    # residual blocks
    for d in range(depth):
        channels = channels * 2
        x = create_downsample(channels, x)
        for b in range(n_blocks):
            x = create_resblock(channels, x)
    # output
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(256, 1, activation=None)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2DTranspose(128, (8,8), (4,4), padding="same", activation=None)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(1, 1, activation='sigmoid')(x)
    outputs = keras.layers.UpSampling2D(2**(depth-2))(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
```

We have the following total param's for the model. Trainable and Non-Trainable param's count is highlighted below.



```
_____________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 80, 80, 128)  2097280     leaky_re_lu_21[0][0]
_____________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 80, 80, 128)  512         conv2d_transpose[0][0]
_____________________________________________________________________________________________
leaky_re_lu_22 (LeakyReLU)      (None, 80, 80, 128)  0           batch_normalization_22[0][0]
_____________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 80, 80, 1)    129         leaky_re_lu_22[0][0]
_____________________________________________________________________________________________
up_sampling2d (UpSampling2D)    (None, 320, 320, 1)  0           conv2d_22[0][0]
=============================================================================================
Total params: 14,957,729
Trainable params: 14,947,297
Non-trainable params: 10,432
_____________________________________________________________________________________________
```

We have reduced the number of epochs to five from our initially planned 15. This was done to reduce the run time. We noticed that each epoch had an ETA of 35 min, averaging upto 7+ hours to complete the training. Due to resource constraint from colab we had decided on this.

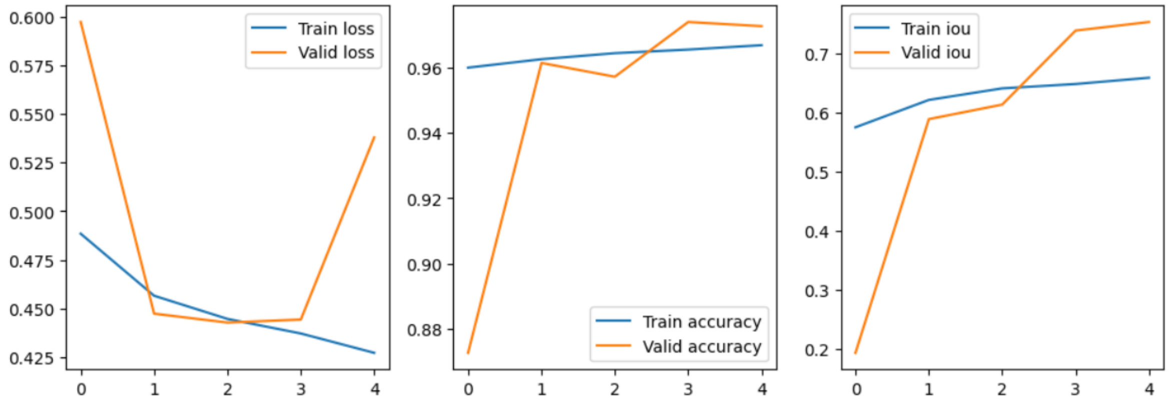


```
history = model.fit(train_gen, validation_data=valid_gen, callbacks=[learning_rate], epochs=5, shuffle=True)

Epoch 1/5
   2/1507 [..............................] - ETA: 4:28 - loss: 0.9098 - accuracy: 0.5511 - mean_iou: 0.0344WARNING:tensorflow:Callba
1507/1507 [==============================] - 2348s 2s/step - loss: 0.4884 - accuracy: 0.9600 - mean_iou: 0.5746 - val_loss: 0.5973 -
Epoch 2/5
1507/1507 [==============================] - 2367s 2s/step - loss: 0.4566 - accuracy: 0.9626 - mean_iou: 0.6210 - val_loss: 0.4473 -
Epoch 3/5
1507/1507 [==============================] - 2436s 2s/step - loss: 0.4446 - accuracy: 0.9644 - mean_iou: 0.6405 - val_loss: 0.4427 -
Epoch 4/5
1507/1507 [==============================] - 2463s 2s/step - loss: 0.4371 - accuracy: 0.9655 - mean_iou: 0.6479 - val_loss: 0.4443 -
Epoch 5/5
1507/1507 [==============================] - 2450s 2s/step - loss: 0.4272 - accuracy: 0.9668 - mean_iou: 0.6585 - val_loss: 0.5380 -
```

We can notice an accuracy of 97.36%. The accuracy of the validation set prediction can be increased at higher epoch's.

Below are the graphical representation of the training of our model.

We can notice big movements on validation loss's, a steady accuracy and steady iou.



Model 2: UNET

We use same class generator from earlier model.

We are increasing the validation set count.

```
n train samples 21684
n valid samples 5000
```

Custom Methods used are:

conv_block (m, dim, acti, bn, res, do=0.5)

level_block (m, dim, depth, inc, acti, do, bn, mp, up, res)

UNet(img_shape, out_ch=1, start_ch=64, depth=5, inc_rate=2., activation='relu',dropout=0.5, batchnorm=True, maxpool=True, upconv=True, residual=True)

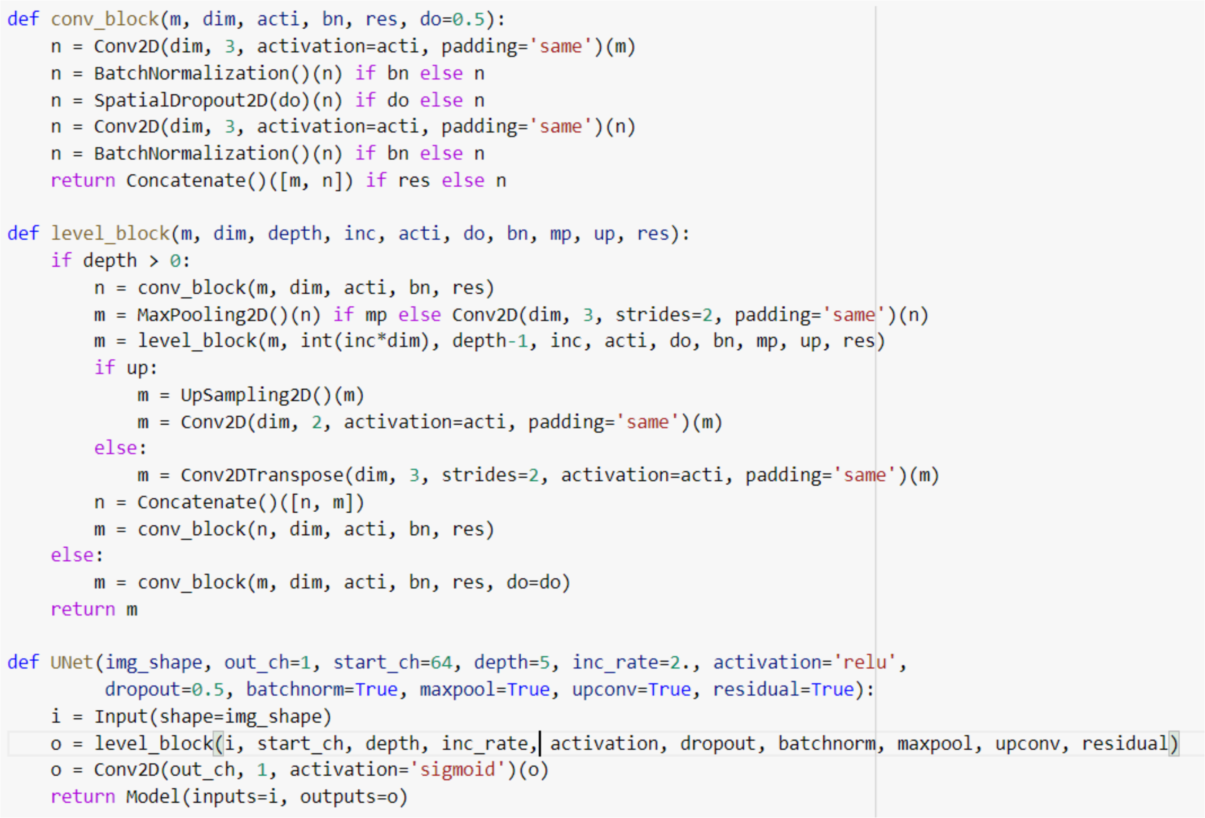
iou_loss (y_true, y_pred)

iou_bce_loss (y_true, y_pred)

mean_iou (y_true, y_pred)

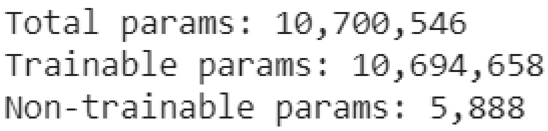
Below are the layers of UNET model that we have decided to use. The same was mentioned in the earlier section.

The core of UNET is defined as below.



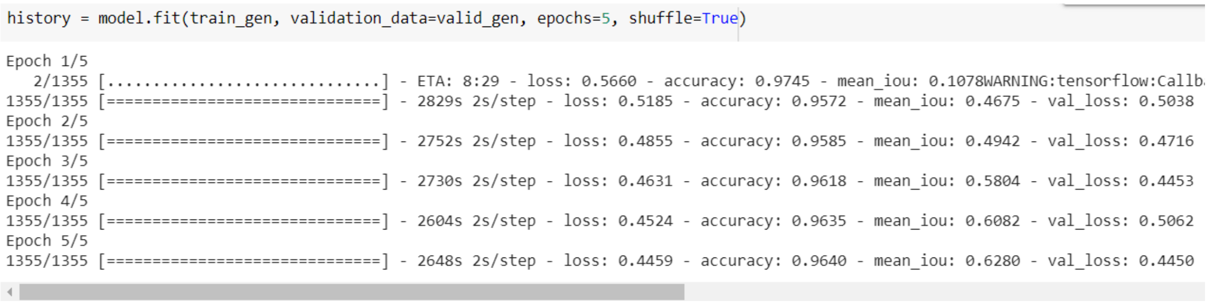
```python
def conv_block(m, dim, acti, bn, res, do=0.5):
    n = Conv2D(dim, 3, activation=acti, padding='same')(m)
    n = BatchNormalization()(n) if bn else n
    n = SpatialDropout2D(do)(n) if do else n
    n = Conv2D(dim, 3, activation=acti, padding='same')(n)
    n = BatchNormalization()(n) if bn else n
    return Concatenate()([m, n]) if res else n

def level_block(m, dim, depth, inc, acti, do, bn, mp, up, res):
    if depth > 0:
        n = conv_block(m, dim, acti, bn, res)
        m = MaxPooling2D()(n) if mp else Conv2D(dim, 3, strides=2, padding='same')(n)
        m = level_block(m, int(inc*dim), depth-1, inc, acti, do, bn, mp, up, res)
        if up:
            m = UpSampling2D()(m)
            m = Conv2D(dim, 2, activation=acti, padding='same')(m)
        else:
            m = Conv2DTranspose(dim, 3, strides=2, activation=acti, padding='same')(m)
        n = Concatenate()([n, m])
        m = conv_block(n, dim, acti, bn, res)
    else:
        m = conv_block(m, dim, acti, bn, res, do=do)
    return m

def UNet(img_shape, out_ch=1, start_ch=64, depth=5, inc_rate=2., activation='relu',
         dropout=0.5, batchnorm=True, maxpool=True, upconv=True, residual=True):
    i = Input(shape=img_shape)
    o = level_block(i, start_ch, depth, inc_rate, activation, dropout, batchnorm, maxpool, upconv, residual)
    o = Conv2D(out_ch, 1, activation='sigmoid')(o)
    return Model(inputs=i, outputs=o)
```

In our first design of UNET we found to have below numbers of parameters.



```
Total params: 10,700,546
Trainable params: 10,694,658
Non-trainable params: 5,888
```

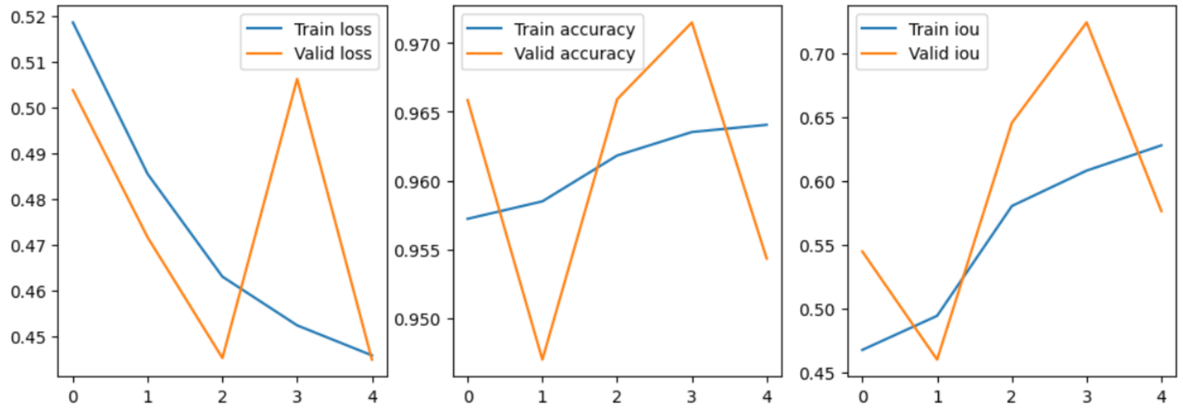
We sticked to 5 epoch’s here as well due to limitations of Google Colab. Also it would take longer runtime. Below one run time summed up to be around 3 hours.



```
history = model.fit(train_gen, validation_data=valid_gen, epochs=5, shuffle=True)

Epoch 1/5
   2/1355 [..............................] - ETA: 8:29 - loss: 0.5660 - accuracy: 0.9745 - mean_iou: 0.1078WARNING:tensorflow:Callb
1355/1355 [==============================] - 2829s 2s/step - loss: 0.5185 - accuracy: 0.9572 - mean_iou: 0.4675 - val_loss: 0.5038
Epoch 2/5
1355/1355 [==============================] - 2752s 2s/step - loss: 0.4855 - accuracy: 0.9585 - mean_iou: 0.4942 - val_loss: 0.4716
Epoch 3/5
1355/1355 [==============================] - 2730s 2s/step - loss: 0.4631 - accuracy: 0.9618 - mean_iou: 0.5804 - val_loss: 0.4453
Epoch 4/5
1355/1355 [==============================] - 2604s 2s/step - loss: 0.4524 - accuracy: 0.9635 - mean_iou: 0.6082 - val_loss: 0.5062
Epoch 5/5
1355/1355 [==============================] - 2648s 2s/step - loss: 0.4459 - accuracy: 0.9640 - mean_iou: 0.6280 - val_loss: 0.4450
```

We found validation accuracy to be at 97.15% which is lesser than RESNET by 0.21%.

But then we tried to hyper tune both these models by varying few parameter which will be discussed on the coming sections.



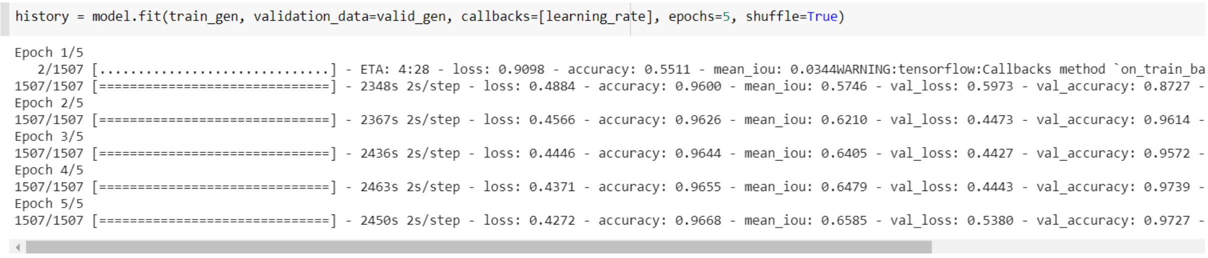
Model evaluation

Our base model remained to be the best of models at 97.39% validation accuracy.

We attempted 4 tried on RESNET and 2 on UNET.

Among all 6 attempts RESNET initial model was found to be the best.

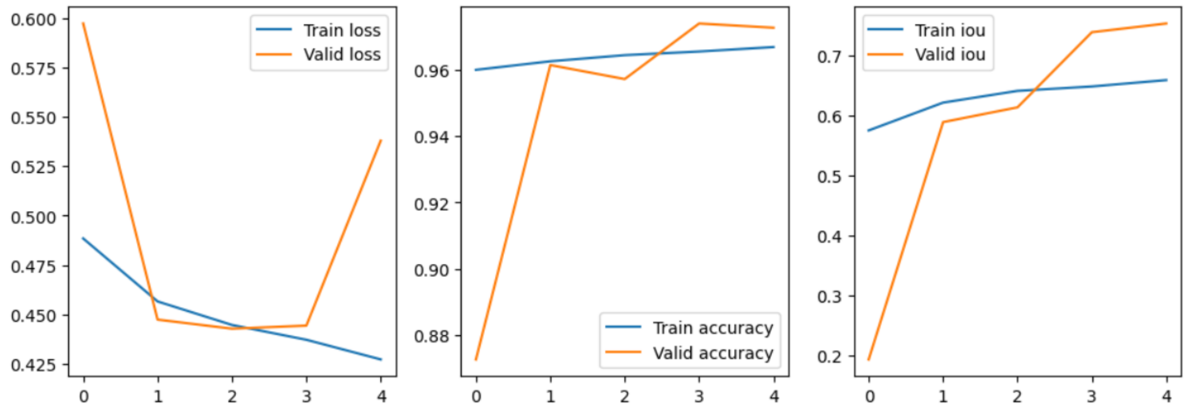
RESNET Base Model



```
history = model.fit(train_gen, validation_data=valid_gen, callbacks=[learning_rate], epochs=5, shuffle=True)
```

```
Epoch 1/5
   2/1507 [..............................] - ETA: 4:28 - loss: 0.9098 - accuracy: 0.5511 - mean_iou: 0.0344WARNING:tensorflow:Callbacks method `on_train_ba
1507/1507 [==============================] - 2348s 2s/step - loss: 0.4884 - accuracy: 0.9600 - mean_iou: 0.5746 - val_loss: 0.5973 - val_accuracy: 0.8727 -
Epoch 2/5
1507/1507 [==============================] - 2367s 2s/step - loss: 0.4566 - accuracy: 0.9626 - mean_iou: 0.6210 - val_loss: 0.4473 - val_accuracy: 0.9614 -
Epoch 3/5
1507/1507 [==============================] - 2436s 2s/step - loss: 0.4446 - accuracy: 0.9644 - mean_iou: 0.6405 - val_loss: 0.4427 - val_accuracy: 0.9572 -
Epoch 4/5
1507/1507 [==============================] - 2463s 2s/step - loss: 0.4371 - accuracy: 0.9655 - mean_iou: 0.6479 - val_loss: 0.4443 - val_accuracy: 0.9739 -
Epoch 5/5
1507/1507 [==============================] - 2450s 2s/step - loss: 0.4272 - accuracy: 0.9668 - mean_iou: 0.6585 - val_loss: 0.5380 - val_accuracy: 0.9727 -
```

We received the best result during the third epoch.



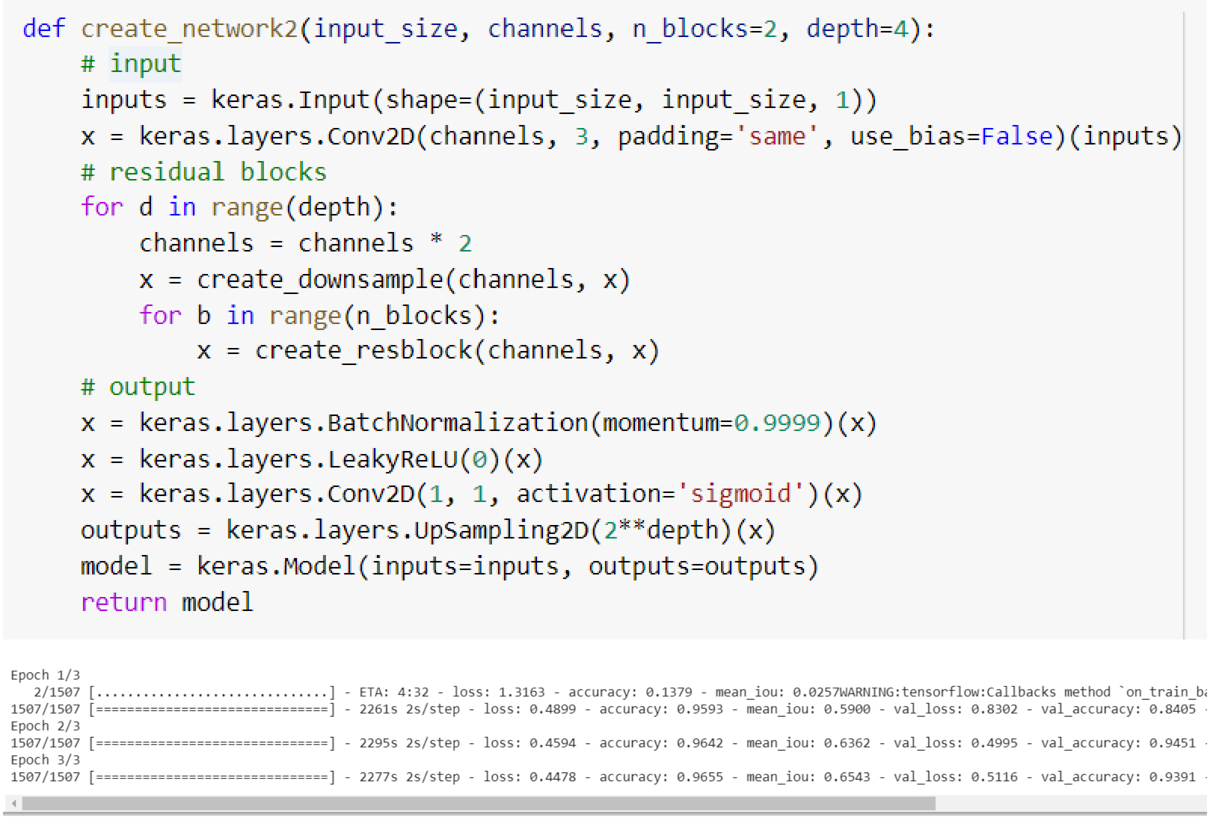
Comparison to benchmark

There was no improvement from our base model. This was due to limited epochs and also reduced layers on RESNET for try 2,3 and 4.

UNET too was ran for just 2 epochs for try 2. Can try few more changes to parameters.

Try 2 RESNET

Reduced layers from base model. Epoch reduced from 5 to 3. Validation accuracy, max at 94.51%



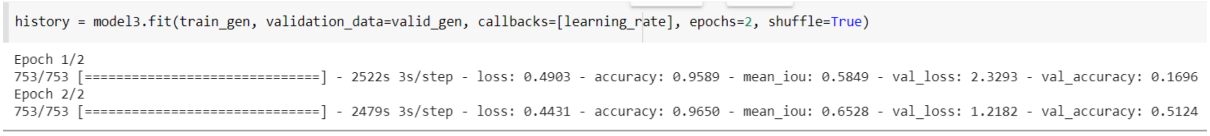
```
def create_network2(input_size, channels, n_blocks=2, depth=4):
    # input
    inputs = keras.Input(shape=(input_size, input_size, 1))
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(inputs)
    # residual blocks
    for d in range(depth):
        channels = channels * 2
        x = create_downsample(channels, x)
        for b in range(n_blocks):
            x = create_resblock(channels, x)
    # output
    x = keras.layers.BatchNormalization(momentum=0.9999)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(1, 1, activation='sigmoid')(x)
    outputs = keras.layers.UpSampling2D(2**depth)(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
```

```
Epoch 1/3
   2/1507 [.............................] - ETA: 4:32 - loss: 1.3163 - accuracy: 0.1379 - mean_iou: 0.0257WARNING:tensorflow:Callbacks method `on_train_ba
1507/1507 [==============================] - 2261s 2s/step - loss: 0.4899 - accuracy: 0.9593 - mean_iou: 0.5900 - val_loss: 0.8302 - val_accuracy: 0.8405 -
Epoch 2/3
1507/1507 [==============================] - 2295s 2s/step - loss: 0.4594 - accuracy: 0.9642 - mean_iou: 0.6362 - val_loss: 0.4995 - val_accuracy: 0.9451 -
Epoch 3/3
1507/1507 [==============================] - 2277s 2s/step - loss: 0.4478 - accuracy: 0.9655 - mean_iou: 0.6543 - val_loss: 0.5116 - val_accuracy: 0.9391 -
```

Try 3 RESNET

Reduced epoch to 2 from 3. Batch size increased to 32 from 16.

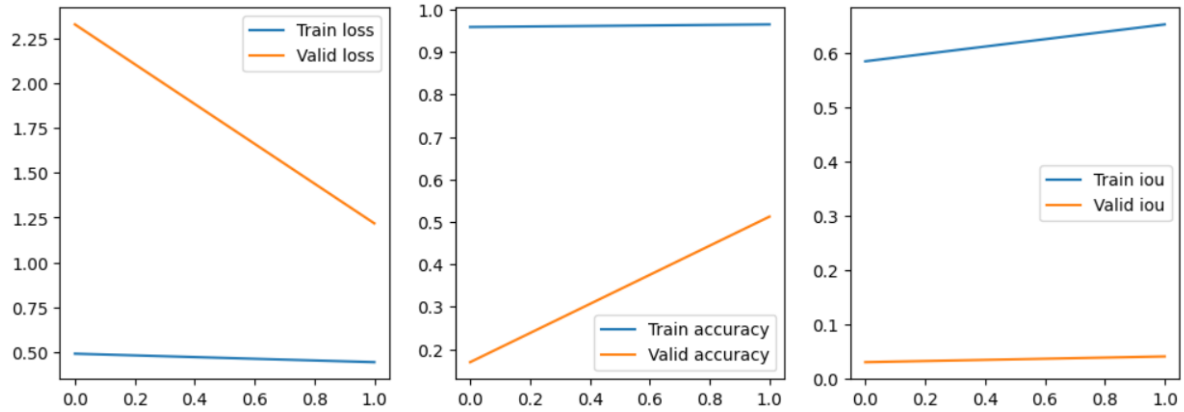
Validation accuracy, max at 51.24%



```
history = model3.fit(train_gen, validation_data=valid_gen, callbacks=[learning_rate], epochs=2, shuffle=True)
```

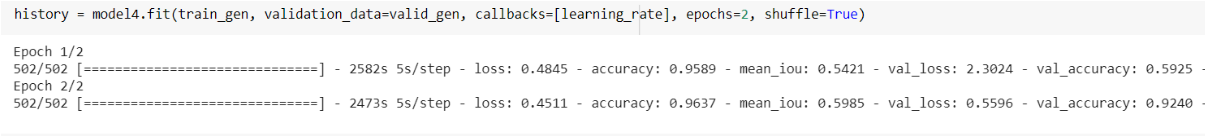
```
Epoch 1/2
753/753 [==============================] - 2522s 3s/step - loss: 0.4903 - accuracy: 0.9589 - mean_iou: 0.5849 - val_loss: 2.3293 - val_accuracy: 0.1696
Epoch 2/2
753/753 [==============================] - 2479s 3s/step - loss: 0.4431 - accuracy: 0.9650 - mean_iou: 0.6528 - val_loss: 1.2182 - val_accuracy: 0.5124
```

,



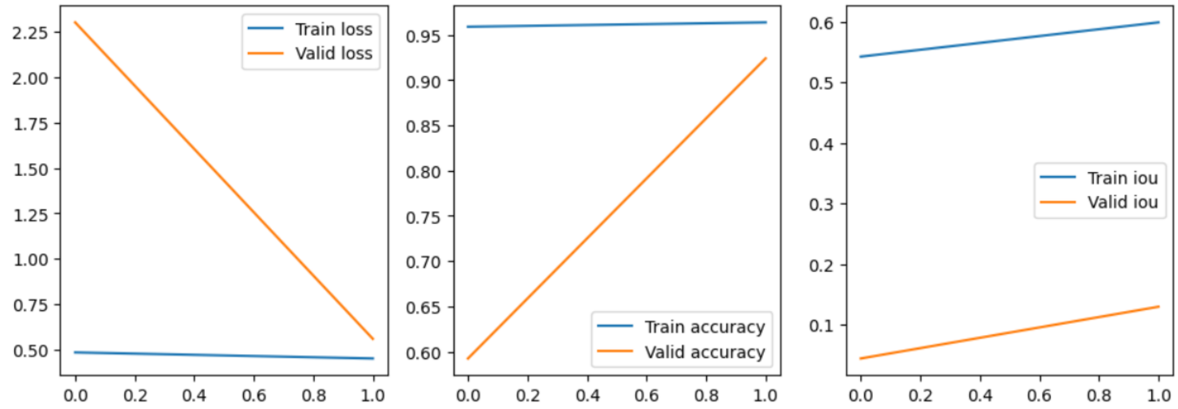
Try 4 RESNET

Changing Image Size and Batch Size to 48 and 256 respectively. Also depth is changed to 3 from 4. Validation accuracy, max at 92.40%



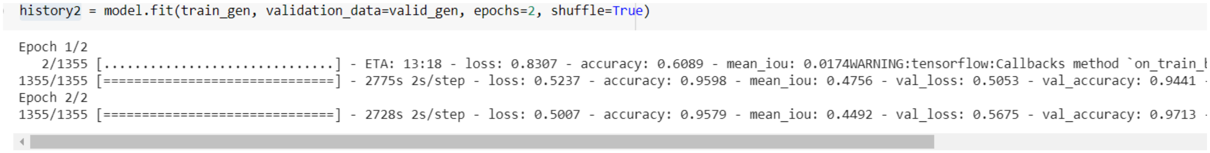
```
history = model4.fit(train_gen, validation_data=valid_gen, callbacks=[learning_rate], epochs=2, shuffle=True)
```

```
Epoch 1/2
502/502 [==============================] - 2582s 5s/step - loss: 0.4845 - accuracy: 0.9589 - mean_iou: 0.5421 - val_loss: 2.3024 - val_accuracy: 0.5925 -
Epoch 2/2
502/502 [==============================] - 2473s 5s/step - loss: 0.4511 - accuracy: 0.9637 - mean_iou: 0.5985 - val_loss: 0.5596 - val_accuracy: 0.9240 -
```



Try 2 UNET

We change the image size to 320 from 256 keeping the batch size same as before. Validation accuracy, max at 97.13%



```
history2 = model.fit(train_gen, validation_data=valid_gen, epochs=2, shuffle=True)
```

```
Epoch 1/2
   2/1355 [..............................] - ETA: 13:18 - loss: 0.8307 - accuracy: 0.6089 - mean_iou: 0.0174WARNING:tensorflow:Callbacks method `on_train_b
1355/1355 [==============================] - 2775s 2s/step - loss: 0.5237 - accuracy: 0.9598 - mean_iou: 0.4756 - val_loss: 0.5053 - val_accuracy: 0.9441 -
Epoch 2/2
1355/1355 [==============================] - 2728s 2s/step - loss: 0.5007 - accuracy: 0.9579 - mean_iou: 0.4492 - val_loss: 0.5675 - val_accuracy: 0.9713 -
```

Implications

Clinicians are faced with reading high volumes of images every shift. Being tired or distracted clinicians can miss important details in image. Here automated image analysis tools can come to help. For example, **one can use machine learning to automate initial detection (imaging screening) of potential pneumonia cases in order to prioritize and expedite their review**. Hence, we decided to develop a model to detect pneumonia from chest radiographs.

Recommendation

Algorithm has high false positive rate, i.e it detects pneumonia in images, where it shouldn't. Hence it would considerably improve the detection accuracy if classified given chest radiograph as "has pneumonia or not" and detect pneumonia symptoms with object detector only if it has pneumonia.

The above model can be used after running at higher epochs and with initial high layers of RESNET with 95%+ confidence level.

Limitations

The problem is, object detectors are good at detecting objects, which have predefined shapes and outlooks. On other hand, opacities in lungs don't have exact shapes. This makes this problem so difficult.

Enhancement for future:

Opacity ('white pixels') exists also outside of lungs in chest radiographs. But we always detect opacities in lungs, because we know that pneumonia is related to problems in lungs. We can let our object detector learn this from the training data or we can help it detecting lungs separately and detect pneumonia in the lungs as a next step.

Closing Reflections

Learning

As the objective is to detect and draw a bounding box on each of the pneumonia opacities, where each image can have 0 or many opacities, and the training set is already classified, it can be analysed as a supervised learning statistical multi-label classification.

For Next time

Definitely try new models such as Yolo and newer versions of RESNET or UNET models.