



AIML

CAPSTONE PROJECT

COMPUTER VISION

PNEUMONIA DETECTION CHALLENGE

FINAL REPORT

By

CAPSTONE-CV1 Proj-Group2

Table of Contents

Background:.....	3
Data	3
Summary of Pre-processing, EDA and Findings.....	4
Findings:.....	4
CSV Files	4
DATA SET - Training.....	5
DATA SET - Testing	5
Pre-processing	6
Merging Dataframe and remove duplicates.....	6
Print random images from merged dataframe along with its class	6
Data balancing	7
Encoding.....	7
EDA	8
Understand Target and Class	8
Insights on coordinates	9
Correlations	9
Visualization of images with coordinates.....	9
Analysis Summary and Next Step.....	10
Milestone – 1 Basic CNN.....	10
Train Test and Validation split	10
Convert to tensor	11
Basic CNN Model	11
Architecture.....	11
Trainable Parameters	11
Accuracy and Loss	12
Accuracy and Recall using dedicated.....	13
Summary and Next step:	13



























Background:

Pneumonia is a Health Condition which is caused by Infection that inflames air sacs in one or both **lungs**, which may fill with fluid. With pneumonia, the air sacs may fill with fluid or pus. The infection can be life-threatening to anyone, but particularly to infants, children and people over 65. This project aims to create a Model using Computer Vision algorithms to detect a visual signal for pneumonia from medical images given as input. The algorithm should provide marker for Lung opacities on the Xray images. The infection in lungs can be in more than one location and algorithm should detect and provide marker for all the inflammation.

Data

Following files are shared for this project,

Shared with me > CV capstone ▾ 🔍

Name	Owner	Last modified	File size
 GCP Credits Request Link - RSNA.txt 	Harish S	Dec 11, 2019 Harish S	55 bytes     
 stage_2_detailed_class_info.csv 	Harish S	Dec 11, 2019 Harish S	1.6 MB 
 stage_2_sample_submission.csv 	Harish S	Dec 11, 2019 Harish S	155 KB 
 stage_2_test_images.zip 	Harish S	Sep 2, 2022 Harish S	378.8 MB 
 stage_2_train_images.zip 	Harish S	Sep 2, 2022 Harish S	3.3 GB 
 stage_2_train_labels.csv 	Harish S	Dec 11, 2019 Harish S	1.4 MB     

1. GCP Credits Request Link - RSNA.txt: The credit file which we don't need to process in the project. It is to give credit to the author of this data.
2. stage_2_detailed_class_info.csv: CSV file having patientid and corresponding class of the disease.
3. stage_2_sample_submission.csv: CSV file which has patientid and predictionstring which is a constant value shown for example. This file may not be required any processing.
4. stage_2_test_images.zip: Zip file containing test images of type DICOM.
5. stage_2_train_images.zip: Zip file containing list of DICOM images which we can use for model training
6. stage_2_train_labels.csv: The CSV File having patientid, coordinates(x, y, width, height) and Target. The target is 0 if there are no coordinates. The Target is 1 if there is a coordinates available.

Summary of Pre-processing, EDA and Findings

Findings:

CSV Files

```
class_info_df = pd.read_csv("CV capstone/stage_2_detailed_class_info.csv")
class_info_df
```

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
...
30222	c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8	Lung Opacity
30223	c1edf42b-5958-47ff-a1e7-4f23d99583ba	Normal
30224	c1f6b555-2eb1-4231-98f6-50a963976431	Normal
30225	c1f7889a-9ea9-4acb-b64c-b737c929599a	Lung Opacity
30226	c1f7889a-9ea9-4acb-b64c-b737c929599a	Lung Opacity

30227 rows × 2 columns

The class info CSV has 30227 records with two column such as patientId and class. There are three classes. They are,

1. No Lung Opacity/Not Normal
2. Normal
3. Lung Opacity

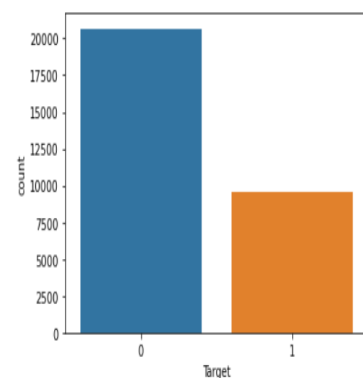
```
train_label_df = pd.read_csv("CV capstone/stage_2_train_labels.csv")
train_label_df
```

	patientId	x	y	width	height	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1
...
30222	c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8	185.0	298.0	228.0	379.0	1
30223	c1edf42b-5958-47ff-a1e7-4f23d99583ba	NaN	NaN	NaN	NaN	0
30224	c1f6b555-2eb1-4231-98f6-50a963976431	NaN	NaN	NaN	NaN	0
30225	c1f7889a-9ea9-4acb-b64c-b737c929599a	570.0	393.0	261.0	345.0	1
30226	c1f7889a-9ea9-4acb-b64c-b737c929599a	233.0	424.0	201.0	356.0	1

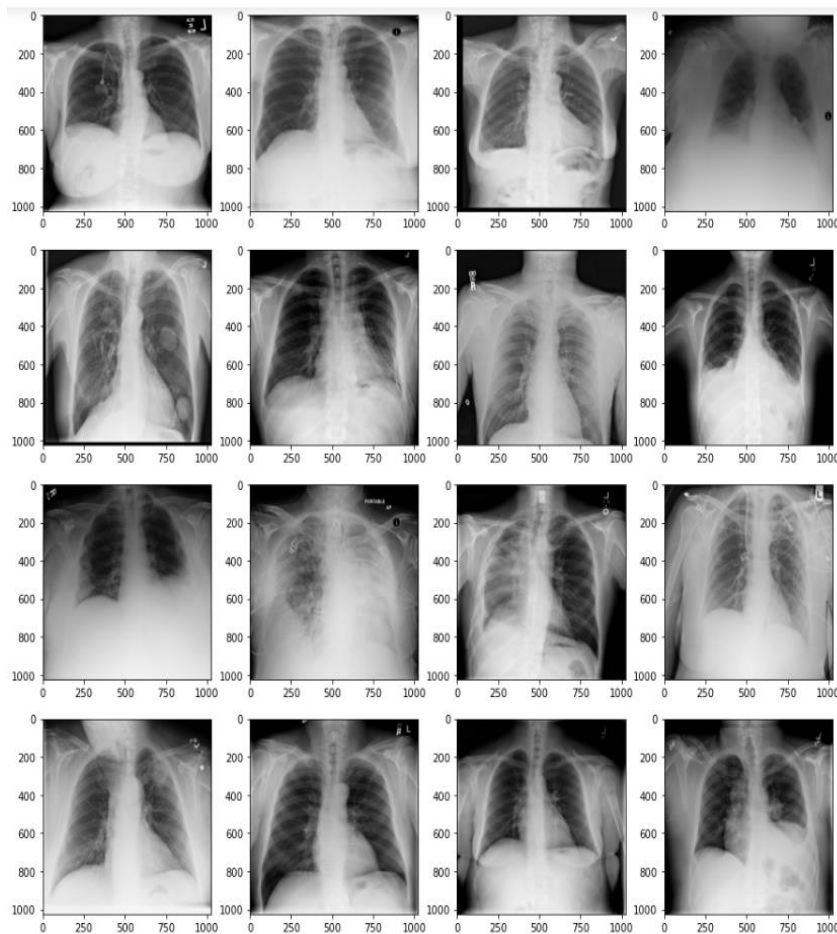
30227 rows × 6 columns

The train label CSV has 30227 records with four coordinate column such as (x, y, width and height) and we have Target Feature and its distribution is,

<AxesSubplot: xlabel='Target', ylabel='count'>



DATA SET- Training



Randomly picked images zipped inside `stage_2_train_images.zip` file. These are DCM images which needs special library such as [pydicom](https://pydicom.github.io/pydicom/) to process. We should install them as it won't come by default. The images can be read and displayed like below,

```
img = dicom.dcmread(img.dcm)
plt.imshow(img)
```

NOTE: We have 26684 image and individual files are named as `patientid.<dcm>`. We will need to pre-process these files as we have more label and class info from CSV, hence remove duplicate if any.

DATA SET- Testing

- We have got 3K images and there is no label and class information details available as they are pure test image set.

Pre-processing

We can load images into pandas dataframe for further processing. While loading we can create attribute of the images such as patientid, image width, height, filename, path and the actual content after resizing images to 32 * 32.

Following line of code can help get the image dataframe.

```
image_directory = zip_directory + "/train_images/stage_2_train_images"
total_number_of_files = 0
image_df = pd.DataFrame(columns=['image_file_name', 'path', 'actual_image', 'height', 'width'])
# Iterate directory
i = 0
for file in os.listdir(image_directory):
    # check if current path is a file
    if os.path.isfile(os.path.join(image_directory, file)):
        total_number_of_files += 1
        dicom_image = dicom.dcmread(os.path.join(image_directory, file))
        image_df.loc[i, 'path'] = os.path.join(image_directory, file)
        image_df.loc[i, 'image_file_name'] = file
        image_df.loc[i, 'actual_image'] = np.array((st.resize(dicom_image.pixel_array, (32, 32), anti_aliasing=True))/255)
        image_df.loc[i, 'height'] = dicom_image.Columns
        image_df.loc[i, 'width'] = dicom_image.Rows
        i+=1
print(f'We have {total_number_of_files} total number of image files');
```

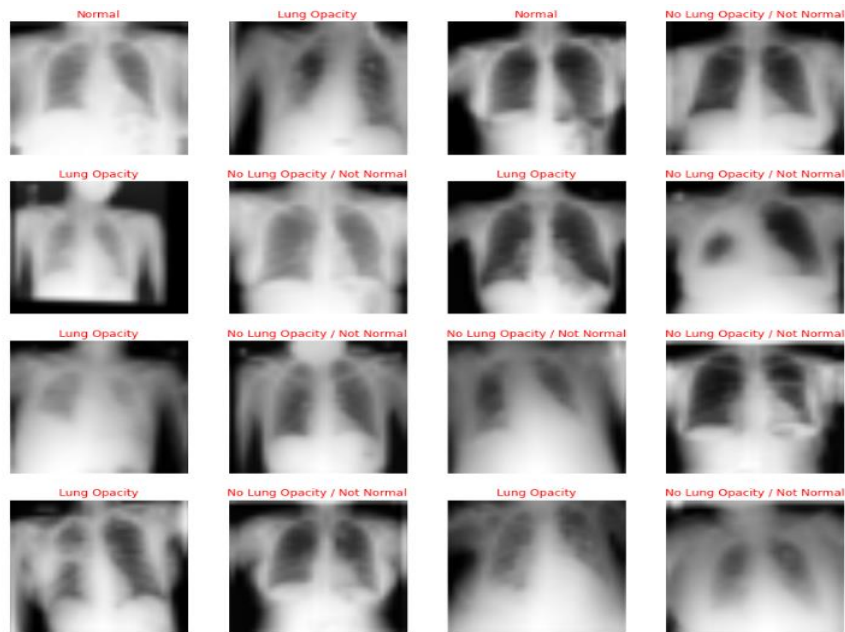
Merging Dataframe and remove duplicates

We can merge image, class and label dataframe to remove duplicate using patientid as common column. After successfully merged three dataframe we are getting 26684 records of image data which will something like below,

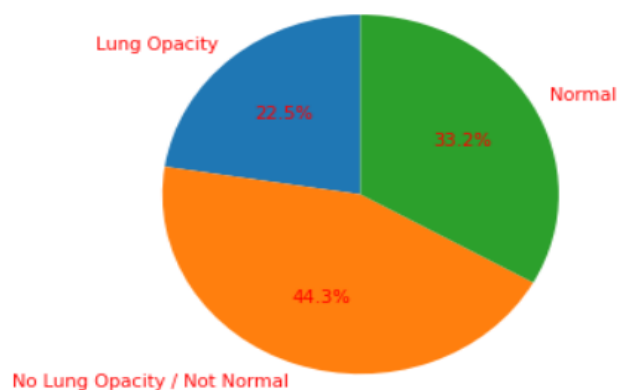
	patientid	image_file_name	path	actual_image	height_x	width_x	extension	x	y	width_y	height_y	Target	class
	0	0004cfab-14fd-4e49-80ba-63a80b6bdd6	/tmp/train_images/stage_2_train_images/0004cfab-14fd-4e49-80ba-63a80b6bdd6.dcm	[[8.286463581287197e-05, 4.041750331910769e-05, ...]]	1024	1024	dcm	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
	1	000924cf-0f8d-42bd-9158-1af53881a557	/tmp/train_images/stage_2_train_images/000924cf-0f8d-42bd-9158-1af53881a557.dcm	[[7.194814904496719e-05, 0.0002037581145740258, ...]]	1024	1024	dcm	NaN	NaN	NaN	NaN	0	Normal
...
26681	fff95b5-805b-4226-80ab-62caec682b22	fff95b5-805b-4226-80ab-62caec682b22.dcm	/tmp/train_images/stage_2_train_images/fff95b5-805b-4226-80ab-62caec682b22.dcm	[[0.0003491565798057978, 0.000770259978522427, ...]]	1024	1024	dcm	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
26682	ffcf11-d018-4414-971a-a7cfa327795	ffcf11-d018-4414-971a-a7cfa327795.dcm	/tmp/train_images/stage_2_train_images/ffcf11-d018-4414-971a-a7cfa327795.dcm	[[1.344063738425597e-05, 4.654971683902636e-07, ...]]	1024	1024	dcm	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
26683	ffec09e-8a4a-48b1-b33e-ab4890ccd136	ffec09e-8a4a-48b1-b33e-ab4890ccd136.dcm	/tmp/train_images/stage_2_train_images/ffec09e-8a4a-48b1-b33e-ab4890ccd136.dcm	[[0.00026488671541695987, 0.000284427266758785, ...]]	1024	1024	dcm	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal

26684 rows × 13 columns

Print random images from merged dataframe along with its class



Data balancing



The class is not perfectly balanced there is a slight imbalance. We will address this data unbalancing in the second part of this project

Encoding

The machine learning or deep learning algorithm requires numbers hence we should convert the class into numbers. We can use LabelEncoder from sklearn preprocessing library like below,

```
# Label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'class'.
data['class_no'] = label_encoder.fit_transform(data['class'])
df = data.drop(labels='class', axis=1)
df
```

EDA

```
df.info()
```

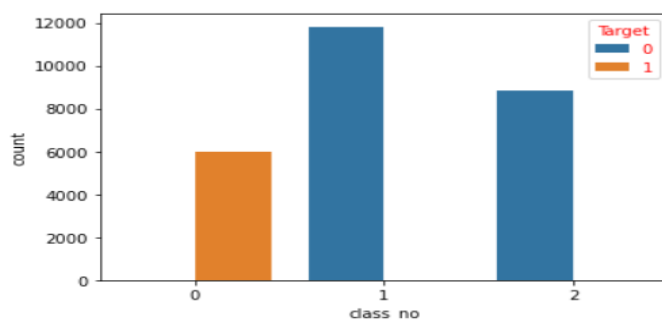
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26684 entries, 0 to 26683
Data columns (total 13 columns):
#   Column              Non-Null Count  Dtype
---  -
0   patientId           26684 non-null  object
1   image_file_name     26684 non-null  object
2   path                26684 non-null  object
3   actual_image        26684 non-null  object
4   height_x            26684 non-null  object
5   width_x             26684 non-null  object
6   extension           26684 non-null  object
7   x                   6012 non-null   float64
8   y                   6012 non-null   float64
9   width_y             6012 non-null   float64
10  height_y            6012 non-null   float64
11  Target              26684 non-null  int64
12  class_no            26684 non-null  int32
dtypes: float64(4), int32(1), int64(1), object(7)
memory usage: 3.8+ MB
```

The dataframe that we have after preprocessing and ready to do EDA. The Target and class_no is a category type

Understand Target and Class

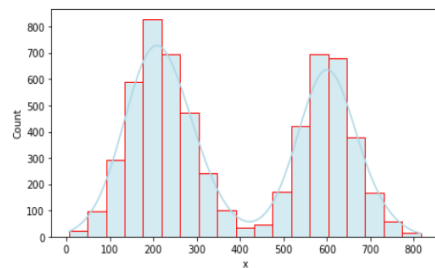
```
sns.countplot(x='class_no',hue='Target',data=df)
```

```
<AxesSubplot:xlabel='class_no', ylabel='count'>
```



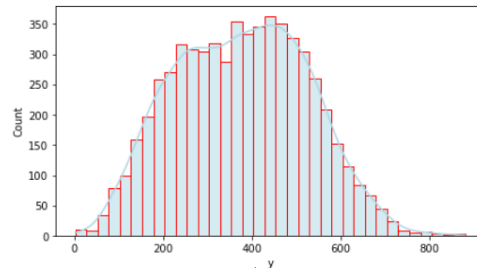
There are 6k records having target as 1 remaining records are set to zero. The zero is nothing but “Lung Opacity” class.

Insights on coordinates



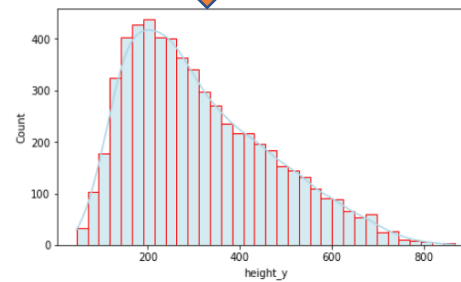
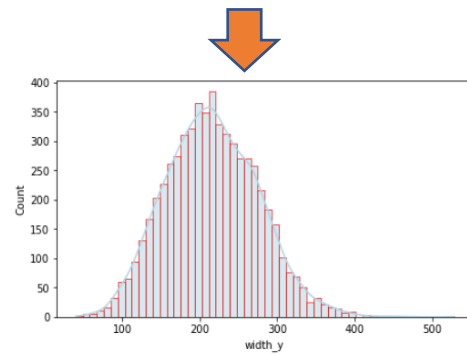
X is range from 0 to 800. There are lot of values are range from 150 to 300 and 550 to 700. We have less count on other ranges

Width_y range from 50 to 400 with few extremes on both side.

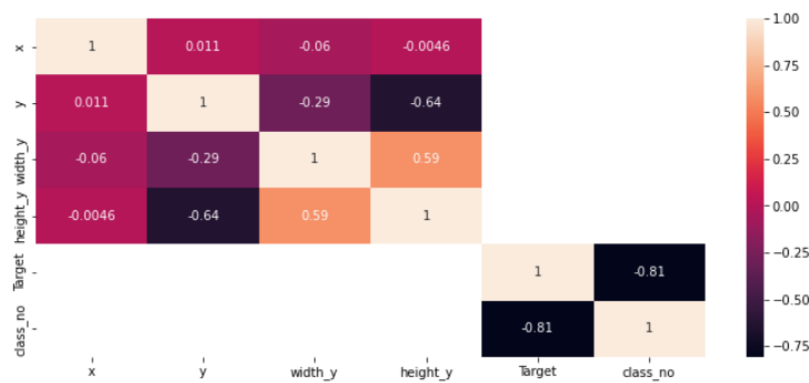


Y is range from 0 to 800 with some extremes. It has lot of records value range from 50 to 450

Height_y is right skewed where we have lot of records having value greater than 200

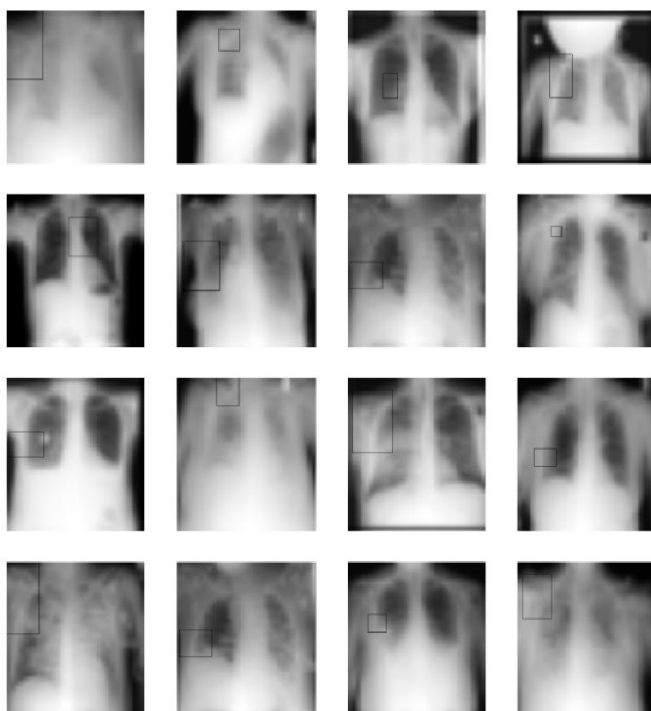


Correlations



We don't have much correlation between variables however average correlation can be seen between Width_y vs Height_y and negative average correlation can be seen between Y and Height_y

Visualization of images with coordinates



```
import cv2
import random

fig = plt.figure(figsize=(15, 15))
N=16
i = 0
#temp_df = df.query("x != nan").sample(n=200)
temp_df = df.loc[df['x'].notnull()]
temp_df = temp_df.sample(100, ignore_index=True)
#print(temp_df)
for i in range(1, N+1):
    loc = random.randint(1, (len(temp_df) - 1))
    fig.add_subplot(4, 4, i)
    x0 = temp_df.loc[loc, 'x'] - temp_df.loc[loc, 'width_y'] / 2
    x1 = temp_df.loc[loc, 'x'] + temp_df.loc[loc, 'width_y'] / 2
    y0 = temp_df.loc[loc, 'y'] - temp_df.loc[loc, 'height_y'] / 2
    y1 = temp_df.loc[loc, 'y'] + temp_df.loc[loc, 'height_y'] / 2
    start_point = (int(x0), int(y0))
    end_point = (int(x1), int(y1))
    plt.imshow(cv2.rectangle(st.resize(temp_df.loc[loc, 'actual_image'], (1024, 1024)), start_point,
                                   end_point, color=(0,0,255), thickness=2), cmap=plt.cm.gray)
plt.axis('off')
plt.show()
```

The above code iterate over the dataframe, reads image array and perform coordinate marking. From the random image, many coordinates comes under right side of the chest xray with very less number of coordinates comes under patient left side.

Analysis Summary and Next Step

We have merged training images with class and training label CSV file. The Analysis suggest that we have 26684 images with 6K records having coordinates and remaining with out coordinates. Since we have images, we should CNN algorithm. We have three classes hence we should use Softmax in the output layer. We will build basic classification model in the first milestone, perform testing and then we will apply down sampling, image augmentation to adjust class balancing, additionally we will use transfer learning technique, Faster RCNN and Mask RCNN in the next phase.

Milestone – 1 Basic CNN

Train Test and Validation split

In order for us to evaluate the model more accurately we will have train, test and validation split. This practise is more nuance. In order to do this split we use following function,

```
def train_test_val_split(X,Y):
    train_ratio = 0.75
    validation_ratio = 0.10
    test_ratio = 0.15

    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=1 - train_ratio)
    x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))
    print(f'{x_train.shape} is x_train shape, {x_test.shape} x_test shape, {x_val.shape} x_val shape, {y_train.shape} is y_train s
    return x_train, x_test, x_val, y_train, y_test, y_val
```

We have our X stored in Dataframe as 'actual_image' and Y as "class_no". Take them out appropriately and split them up.

Convert to tensor

The CNN models will take image of size (N,width,height,RGB). We need to convert our actual_image array to tensor and following function can help you achieve that.

```
from skimage.color import gray2rgb

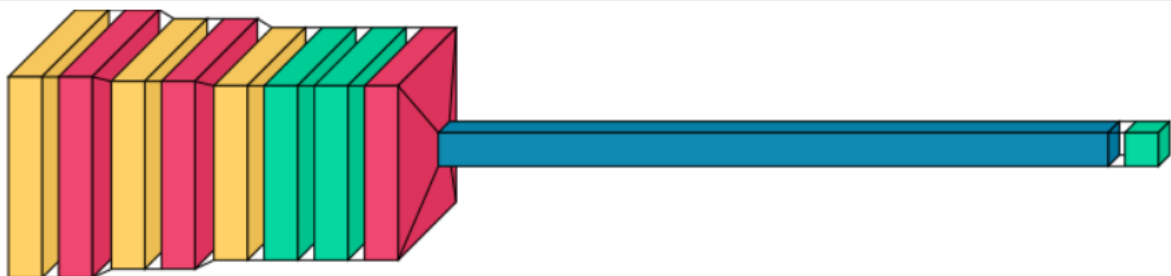
X_train = gray2rgb(x_train.to_list())
X_test = gray2rgb(x_test.to_list())
X_val = gray2rgb(x_val.to_list())

print(f'{X_train.shape} is X_train shape, {X_test.shape} X_test shape, {X_val.shape} is X_val shape, {y_train.shape} is y_train s
(20013, 32, 32, 3) is X_train shape, (4003, 32, 32, 3) X_test shape, (2668, 32, 32, 3) is X_val shape, (20013,) is y_train shap
e, (4003,) y_test shape, (2668,) y_val shape
```

Basic CNN Model

Architecture

3 Conv Layer, 3 Batch Normalization, 2 Dense, 1 Flatten and Output layer with softmax.

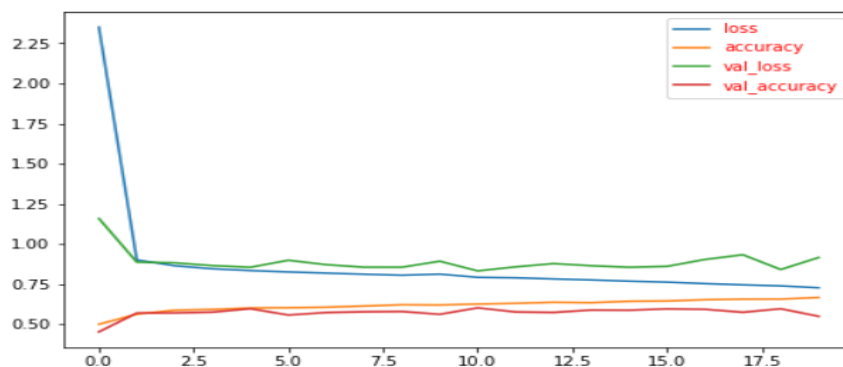


Trainable Parameters

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 64)	1792
batch_normalization_3 (Batch Normalization)	(None, 30, 30, 64)	256
conv2d_4 (Conv2D)	(None, 28, 28, 64)	36928
batch_normalization_4 (Batch Normalization)	(None, 28, 28, 64)	256
conv2d_5 (Conv2D)	(None, 26, 26, 128)	73856
dense_3 (Dense)	(None, 26, 26, 128)	16512
dense_4 (Dense)	(None, 26, 26, 64)	8256
batch_normalization_5 (Batch Normalization)	(None, 26, 26, 64)	256
flatten_1 (Flatten)	(None, 43264)	0
dense_5 (Dense)	(None, 3)	129795
Total params: 267,907		
Trainable params: 267,523		
Non-trainable params: 384		

Accuracy and Loss

```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.show()
```



The accuracy of training is showing small improvements and testing was little going up and down.

Accuracy and Recall using dedicated

We have a validation data available with use which we can use to run against the model to accurately understand the accuracy, recall and precision. We can do this using following code,

```
y_val_prediction_model = model.predict(X_val)
y_val_prediction=[]
for i in y_val_prediction_model:
    y_val_prediction.append(np.argmax(i))

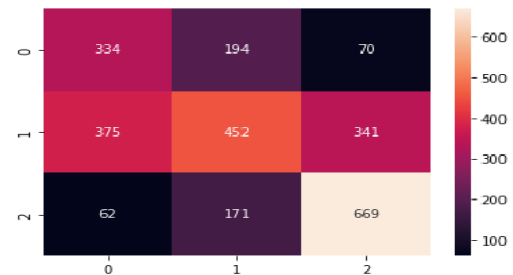
cr = classification_report(y_val,y_val_prediction)
print(cr)

#classification_report(label_encoder.inverse_transform(y_v
```

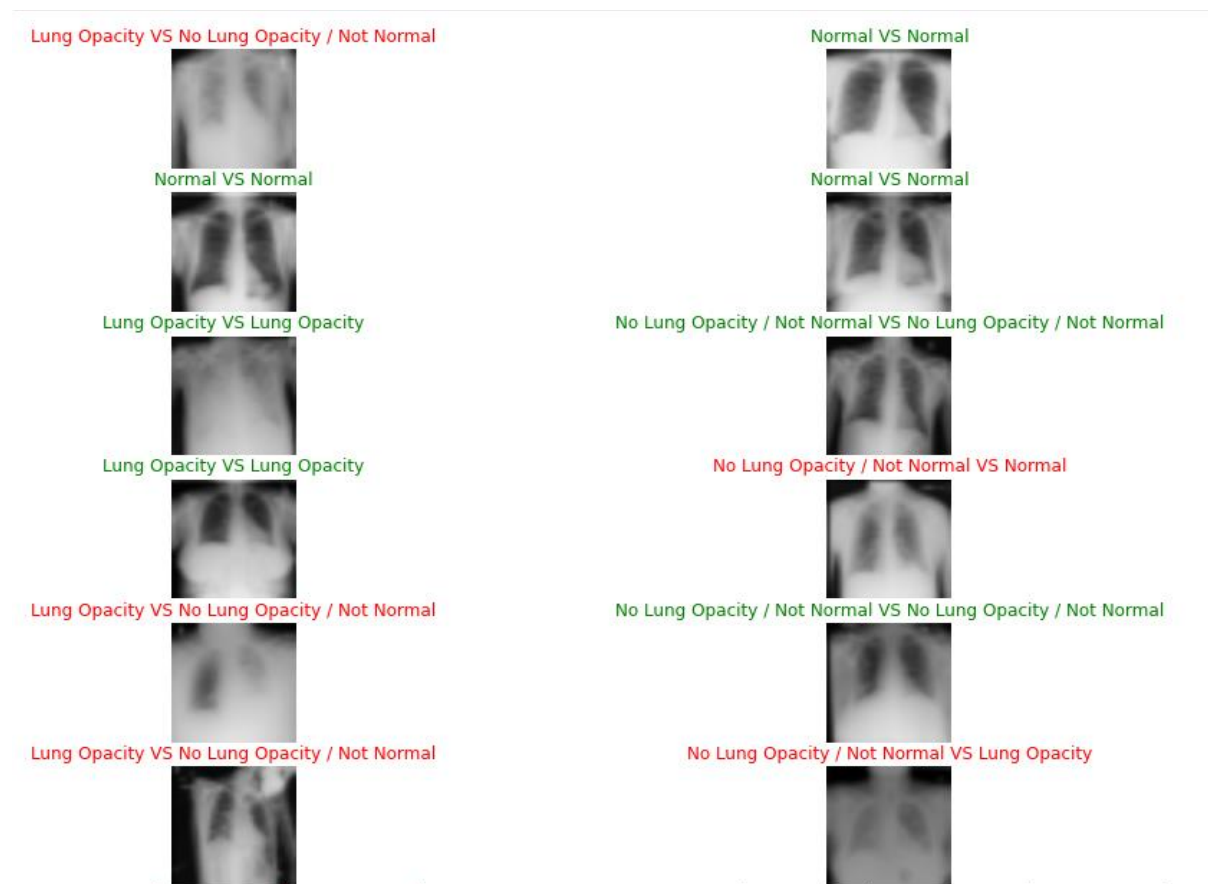
```
84/84 [-----] - 8s 86ms/step
              precision    recall  f1-score   support

     0         0.43       0.56       0.49         598
     1         0.55       0.39       0.46        1168
     2         0.62       0.74       0.68         902

 accuracy          0.54
 macro avg          0.54
 weighted avg       0.54
```



We can also get actual classification vs predicted classification like below which shows lot of images are misclassified to different class (Red color)



Next step:

The basic model we have got to do classification is giving low recall on predicting lung opacity and for normal class as low as well. We will be doing further optimization for this model like Image Augmentation, Parameter tuning such updating learning rate, trying with Adam Optimizer, transfer learning optimization and trying with more convolution layer, etc in the second phase. Later we will implement Object detection using Faster RCNN and Mask RCNN, etc.

Fine Tuning Classification Model:

Following are the approach we are going to try to fine tune the model we had to get more accuracy,

1. Down size the data and apply the CNN model
2. Hyper parameter tuning
3. Apply Following Transfer Learning
 - a. VGG16
 - b. ResNet50
 - c. DenseNet169
 - d. MobileNet

DownSize data:

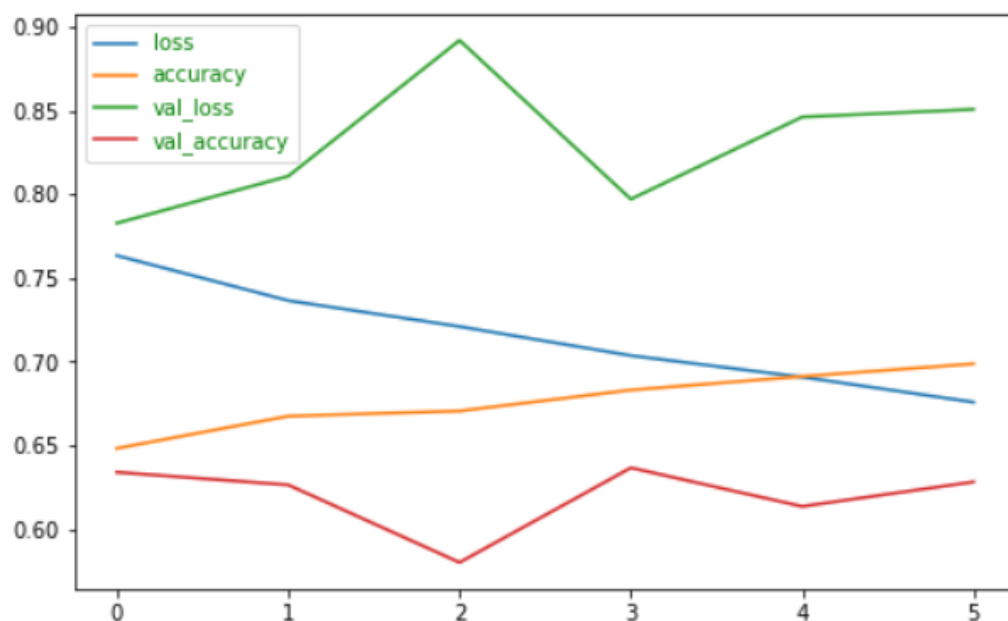
In order to downsample the data, apply lower denominator threshold like below in each class and use the resulting sample for training and validation

```
threshold = 6012
df_0 = df.query('(class_no == 0)').sample(n=(threshold))
df_1 = df.query('(class_no == 1)').sample(n=(threshold))
df_2 = df.query('(class_no == 2)').sample(n=(threshold))

down_df = pd.concat([df_0.sample(frac=1), df_1.sample(frac=1), df_2.sample(frac=1)], ignore_index=True)
down_df.shape

(18036, 13)
```

Following is the result of the training using the previous model,



With downsampled data - we are seeing similar trend. Accuracy is not improving.

Increase Hidden Layer

We then added more hidden layer using following code on the previous model,

```
model2 = Sequential()
model2.add(InputLayer(input_shape = (32,32,3)))
for layer in model.layers[:-3]:
    model2.add(layer)
model2.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model2.add(Dropout(0.2))
model2.add(BatchNormalization())
model2.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.2))
model2.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.2))
model2.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model2.add(BatchNormalization())
model2.add(Flatten())
model2.add(Dense(3, activation='softmax'))
model2.build()
```

With the additional layers in the existing model when we performed test on the original data we get following result which shows clear improvement of accuracy and reduction of loss from previous model,

```

r = np.arange(2)
width = 0.25
bar1 = plt.bar(r, model1_eval_result, color = 'r',
               width = width, edgecolor = 'black',
               label='Model1')
bar2 = plt.bar(r + width, model2_eval_result, color = 'g',
               width = width, edgecolor = 'black',
               label='Model2')

plt.legend()
plt.title("Loss and Accuracy plot for Model 1 and 2")
plt.show()

```



Testing accuracy is 66% which is higher with the addition of 4 more hidden layers in Model 2

Hyper Parameter tuning

We then tried to find the better learning rate, optimum and used Adam optimizer using following code but due to hardware/time limitation – we could not run for all possible combinations. With the standard values suggested over internet we did not see improvements.

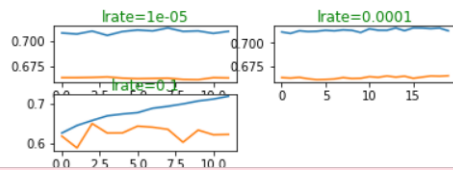

```
def fit_model_find_lr(trainX, trainy, testX, testy, lrate):
    optimizer = optimizers.SGD(learning_rate=lrate)
    model3.compile(loss = 'sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    earlyStoppingCallback = EarlyStopping(monitor='val_loss', patience=5)
    model3_history = model3.fit(X_train, y_train, epochs = 10, batch_size=100,
                               validation_data=(X_test, y_test), callbacks=[earlyStoppingCallback], verbose=0)

    # plot Learning curves
    plt.plot(model3_history.history['accuracy'], label='train')
    plt.plot(model3_history.history['val_accuracy'], label='test')
    plt.title('lr='+str(lrate), pad=-50)

learning_rates = [0.00001, 0.0001, 0.1]
for i in tqdm(range(len(learning_rates))):
    plot_no = 420 + (i+1)
    plt.subplot(plot_no)
    plt.tight_layout()
    fit_model_find_lr(X_train, y_train, X_test, y_test, learning_rates[i])

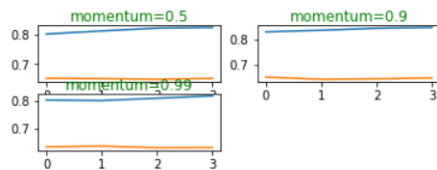
# show learning curves
plt.show()
```

100% | 3/3 [6:09:30<00:00, 7390.12s/it]



Description: Code to find best LR

With learning rate 0.1 we seem to be hitting some thing above 60% accuracy. Earlier with default learning rate we have had 65% accuracy on test data. So we will get some good accuracy when we have learning rate $lr > 0.001$ and < 0.1 . We should try something in between. Due to resource limitation on personal computer we will stick with 0.001 which is default LR for SGD.



With momentum 0.99 there is a small upward trend but it sounds like overfitting. It appears that default momentum is better. If we run for more epoch in good hardware we may be able to find better result

Description: Testing Momentum

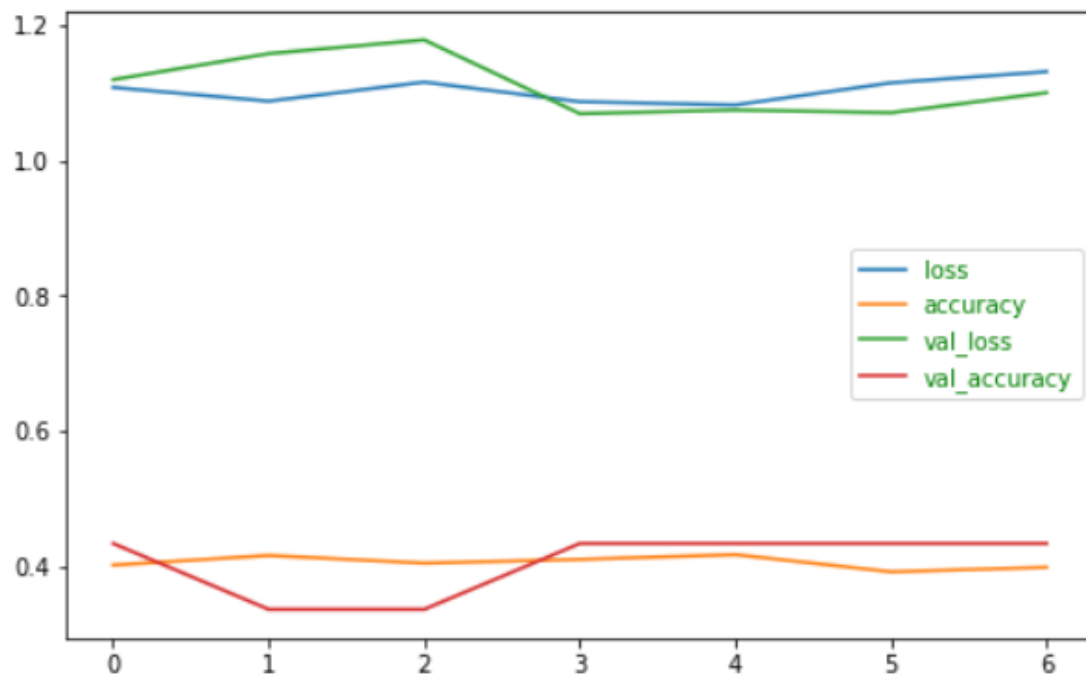
We tried Adam Optimizer which resulted in overfitting,

```
model4.compile(optimizer='Adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
earlyStoppingCallback = EarlyStopping(monitor='val_loss', patience=3)
model4_history = model4.fit(X_train, y_train, epochs = 10, batch_size=BATCH_SIZE,
                            validation_data=(X_test, y_test), callbacks=[earlyStoppingCallback])
```

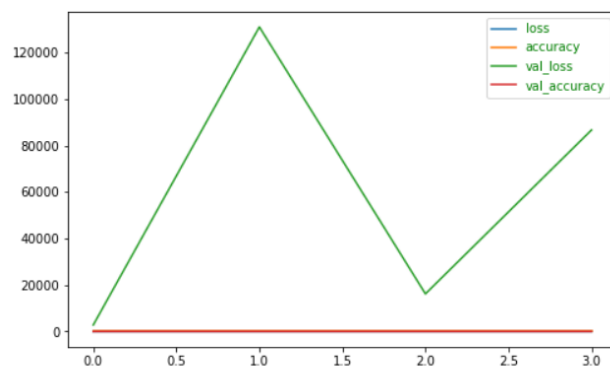
Epoch 1/10
626/626 [=====] - 919s 1s/step - loss: 0.7462 - accuracy: 0.6574 - val_loss: 71832.7344 - val_accuracy: 0.3362
Epoch 2/10
626/626 [=====] - 911s 1s/step - loss: 0.7033 - accuracy: 0.6780 - val_loss: 5490.9761 - val_accuracy: 0.3362
Epoch 3/10
626/626 [=====] - 943s 2s/step - loss: 0.6704 - accuracy: 0.6948 - val_loss: 9625.1533 - val_accuracy: 0.3362
Epoch 4/10
626/626 [=====] - 986s 2s/step - loss: 0.6517 - accuracy: 0.7050 - val_loss: 285.7574 - val_accuracy: 0.3362
Epoch 5/10
626/626 [=====] - 954s 2s/step - loss: 0.6379 - accuracy: 0.7112 - val_loss: 4505.6338 - val_accuracy: 0.2301
Epoch 6/10
626/626 [=====] - 1017s 2s/step - loss: 0.6202 - accuracy: 0.7208 - val_loss: 2333.2161 - val_accuracy: 0.2301
Epoch 7/10
626/626 [=====] - 993s 2s/step - loss: 0.6045 - accuracy: 0.7296 - val_loss: 13308.5488 - val_accuracy: 0.2301

Transfer Learning

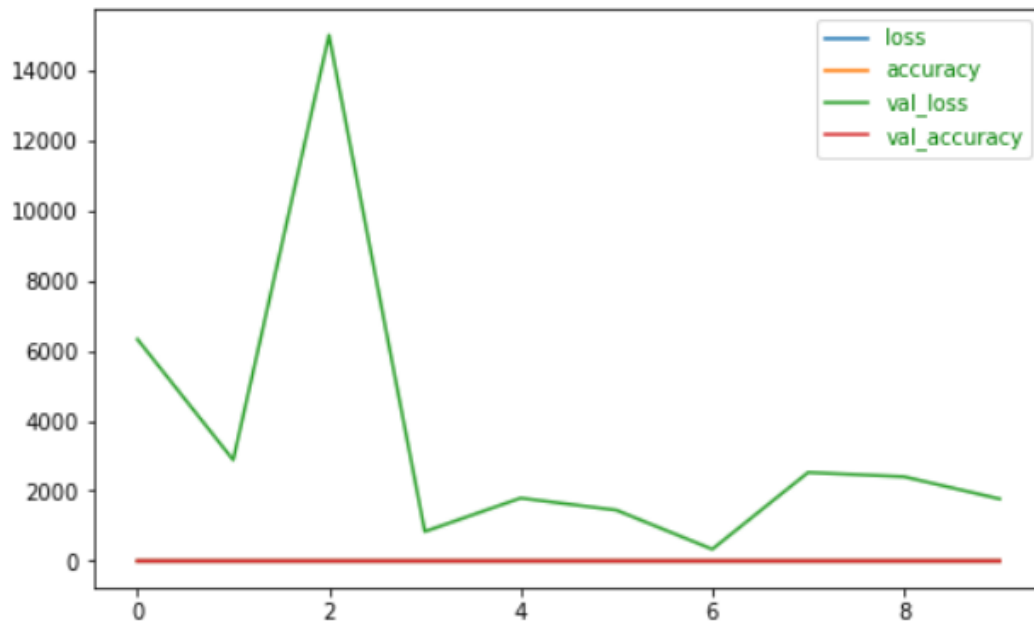
VGG16 : Accuracy is very poor



ResNet50 and DenseNet: It was showing very high loss and a clear indication for overfitting



Description RestNet Result



Description: DesnseNet Result

Classification Summary

	model_name	loss	accuracy	description
0	model_1	1.238485	0.502123	Basic CNN with 3 hidden layer without dropout
1	model_2	0.744067	0.662004	Basic CNN with 7 hidden layer, dropout
2	model_3	0.000000	0.000000	Refer Step - 1.2.1
3	model_4	15340.288544	0.290746	Model2 with Adam optimizer
4	vgg_model_default	NaN	0.230077	Vegg Model
5	vgg_lr_mom	1.109797	0.405838	Vegg Model with LR as 0.1 and Mom as .99
6	ResNet50_default	59150.973267	0.360605	ResNet50
7	DenseNet169	3531.678098	0.365476	DenseNet169 Default
8	DenseNet169_downsample	0.954320	0.521890	DenseNet169 Default with downsample data
9	DenseNet169_downsample_epoch	0.954320	0.521890	DenseNet169 Default with downsample data with ...
10	DenseNet169_downsample_epoch_mom	0.853227	0.592672	DenseNet169 Default with downsample data with ...
11	MobileNet_SDG	1.624691	0.352486	MobileNet with SDG Optimizer
12	MobileNet_Adam	27.065041	0.336248	MobileNet with Adam Optimizer

The above table clearly indicates that our model with 7 hidden layer is producing better accuracy however this is something not acceptable. In order to try with more hidden layer and to try more optimization parameters we can use packages like KerasTuner. This also means that we need more dedicated system available 24/7 (kind of commodity server). With the PC's we could not run anything more further.

Object Detection

This is a main puzzle which we want to solve however due to project time limit we could not complete this. Following is what we would like to try,

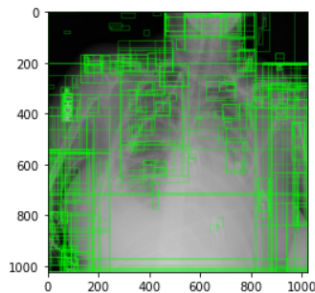
RCNN:

1. It works based on Selective Search

Following results were obtained on one image to see the clustering,

```
: input_image = cv2.imread(os.path.join(TRAIN_IMAGE_FOLDER, "000db696-cf54-4385-b10b-6b16fbb3f985.jpg"))
ss_object.setBaseImage(input_image)
ss_object.switchToSelectiveSearchFast() #this method of createSelectiveSearchSegmentation()
rects = ss_object.process() # The output of the process is a set of a potential ROI's, depending on the size of the base image
new_input_image = input_image.copy() # create copy of the base image
for i, rect in enumerate(rects):
    x, y, w, h = rect
    cv2.rectangle(new_input_image, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
# plt.figure()
plt.imshow(new_input_image)
```

: <matplotlib.image.AxesImage at 0x25c8c9004f0>



Mask RCNN:

<TODO>

SUMMARY

We got zip consists of image and CSV files which was clearly preprocessed and performed EDA on the dataset. Later we tried basic CNN and did classification and then improved performance accuracy for classification using additional hidden layers. We also tried transfer learning for improvements however we could not better results with transfer learning. It is also possible that we can still do more hyper parameter tuning on transfer learning, etc. However due to time, hardware and resource limitations we could not try anything more. Finally the object detection is something is not tried to the full extent due to time limit.