Introduction

Pneumonia is when one or both of a patient's lungs tissue becomes swollen which is caused by a bacterial infection to the lungs.

People over the age of 65 and under the age of 2 are most at risk of pneumonia affecting them the most due to their weakened immune system. Chest x rays are the best way of identifying weather or not a patient has pneumonia. They will do this by looking to see if there are any small white spots on the lungs which is the most tell tail sign of pneumonia.

Deep learning can be applied to images of chest x rays to try and determine whether or not a patient has pneumonia, this can be very useful as it can free up doctors time to do other things and can also lead to a higher accuracy of detection as can look more in depth at the images., The more a deep learning algorithm studies images and understand patterns, the more accurate it can become and help doctors to identify the disease. Both a baseline CNN and two pre-defined transfer models have been used on a set of test images to check what the accuracy the model can achieve when identify pneumonia.

https://www.nhs.uk/conditions/pneumonia/

https://www.radiologyinfo.org/en/info/pneumonia#:~:text=Chest%20x%2Dray%3A%20An%20x,infiltrates)%20tha

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0256630

1 Research & Data Exploration

1.1 Research

https://ieeexplore.ieee.org/document/9591581

The paper looks to investigate if using deep learning models would help to improve the accuracy of diagnosing pneumonia. As of now, a radiologist is required to use their judgment when looking at chest x-rays (CXR) to decide whether they think that the patient has pneumonia or not. This can cause issues as it can be very time-consuming and lead to a possible human error occurring. The researcher has used an automatic pneumonia detection system that uses three different deep learning models to try and predict the likelihood of pneumonia. In carrying out their research the researcher managed to have a model which was highly effective in getting a high accuracy score of 98.32% for predicting Pneumonia which is a vast improvement over human judgment. The paper displays the results from their accuracy results through charts comparing the training data and validation data. It is very easy to interpret what the model accuracies are and how well the models have been able to predict pneumonia. They display the data in multiple ways to help you get a full understanding of the results. The researcher does well in showing the effect of using CLAHE (used to enhance the images and uniform them all to being 224 x 224) and PCA (used to reduce the dimensions of a dataset from 512 features to 100 features) on the images before putting them through the CNN model. This helps to show why it is important to do some image pre-processing prior to running the model as it helps to improve the accuracy of predicting pneumonia and through tables and charts the researcher has displayed this very clearly. The researcher could have included some of their tests that produced worse outcomes to give more of a strong reason for why they used the model which they did. There is not a strong justification for the choices they have made for their CNN model, they do explain why they have done each step but do not give a reason why they haven't used more or less of a certain part. It would be very useful to see these to help improve the validity of the model. The paper overall does very well in investigating using CNN models to predict the likelihood of pneumonia being present. The researcher has managed to develop a system with an extremely high accuracy which in turn could help greatly in hospitals as it will help predict pneumonia with much higher accuracy and free up doctors' time as they will not have to waste time investing the images themselves. Between the CNN model and the doctor, patients have a greatly heightened chance of diagnosis, which could save patients' lives.

1.2 Data Exploration

CNN and Transfor Learning CW

In [3]:

```
#Mounting google drive to colab notebook
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [4]:

```
#Providing the config path to kaggle.json
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Kaggle"
# /content/gdrive/My Drive/Kaggle is the path where kaggle.json is present in the Google
Drive
```

In [5]:

```
#changing the working directory
%cd /content/gdrive/My Drive/Kaggle
#Check the present working directory using pwd command
```

```
/content/gdrive/My Drive/Kaggle
```

In [6]:

```
#Api kaggle command
!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
```

```
chest-xray-pneumonia.zip: Skipping, found more recently modified local copy (use --force
to force download)
```

Building own CNN

In [7]:

```
# import the required libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import seaborn as sns
from matplotlib.image import imread
from PIL import Image
import tensorflow as tf
from tensorflow.keras import layers
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout,AveragePooling2D,Flatten, Dense,
Conv2D,MaxPool2D, MaxPooling2D,BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
import warnings


warnings.filterwarnings('ignore')
```

Load and Explore Data

In [8]:

```
# data stored in the local drive
main_dirction_path = 'chest_xray'
print(os.listdir(main_dirction_path))
```

```
['__MACOSX', 'chest_xray', 'test', 'train', 'val']
```

In [9]:

```
# path for train/ test and validate folders
train_folder_path = main_dirction_path + '/train/'
test_folder_path = main_dirction_path + '/test/'
val_folder_path = main_dirction_path + '/val/'
```

The image size has been set to 90X90 for this model due to studies showing that for accurate image classification a high image size has been set. The study shows that the highest possible resolution should be used for the most accuirate response. Usually, the higher the image resolution the more accurate the model becomes but due to memory constraints the image quality cannot be any higher than this. 90x90 gives the best trade-off between quality and memory constraints.

I have tested with a range of image sizes and it turens out that 90 actually helps to incrase the accuracy, this could be due to the image augmentation i am also doing to enhance the images and with a larger image size could strech the images.

https://pubs.rsna.org/doi/full/10.1148/ryai.2019190015

In [10]:

```
labels = os.listdir(train_folder_path)
img_size = 90
```

In [11]:

```
train_n_path = train_folder_path+'/NORMAL/'
train_p_path = train_folder_path+'/PNEUMONIA/'
test_n_path = test_folder_path+'/NORMAL/'
test_p_path = test_folder_path+'/PNEUMONIA/'
```

In [12]:

```
print(f'Number of normal images is {len(os.listdir(train_n_path))}') #length of normal tr
aining images
print(f'Number of postive images is {len(os.listdir(train_p_path))}') #length of pneumoni
a training images
print(f'Total training images is {len(os.listdir(train_n_path)) + len(os.listdir(train_p_
path))}')
```

```
Number of normal images is 1341
Number of postive images is 3875
Total training images is 5216
```

This allows me to see how many images are present in the training set of images.

In [13]:

```
def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img))
                resized_arr = cv2.resize(img_arr, (img_size, img_size)) #Reshaping image
s to preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
              print(e)
    return np.array(data)
```

In [14]:

```
# Load the datasets into the train, test and validation variables
train = get_training_data(train_folder_path)
```

```python
test = get_training_data(test_folder_path)
val = get_training_data(val_folder_path)
```

In [15]:

```python
print(f'The shape of the training set is {train.shape}')
```

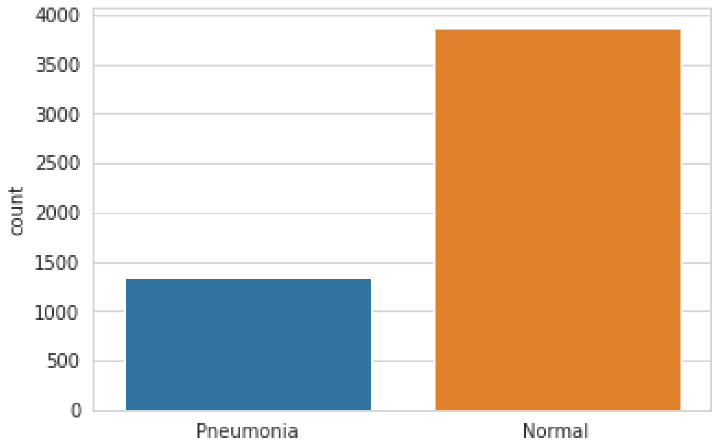
```
The shape of the training set is (5216, 2)
```

In [16]:

```python
l = []
for i in train:
    if(i[1] == 0):
      l.append("Pneumonia")
    else:
      l.append("Normal")
sns.set_style('whitegrid')
sns.countplot(l)
```

Out[16]:

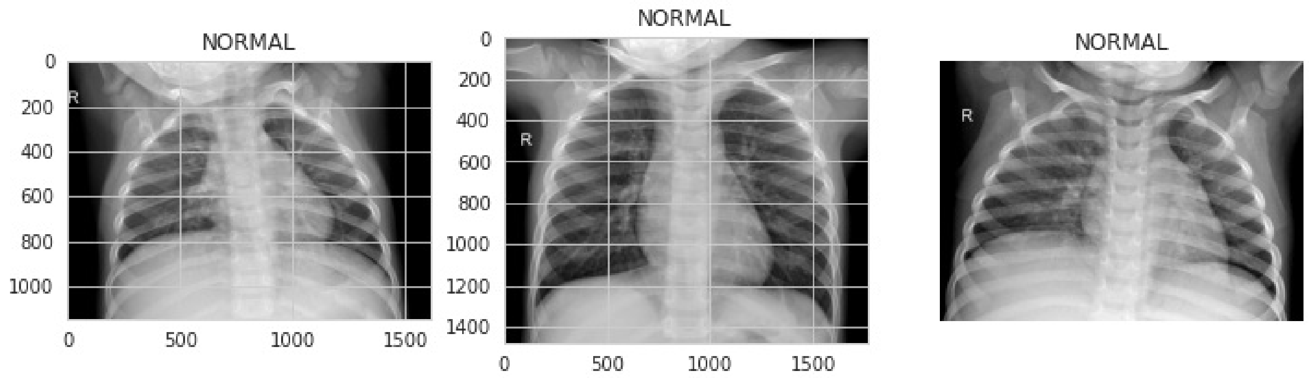
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1498d7bb10>
```



Below is a function to show arbitray number of normal or pneumonia images subject to the arguments passed.

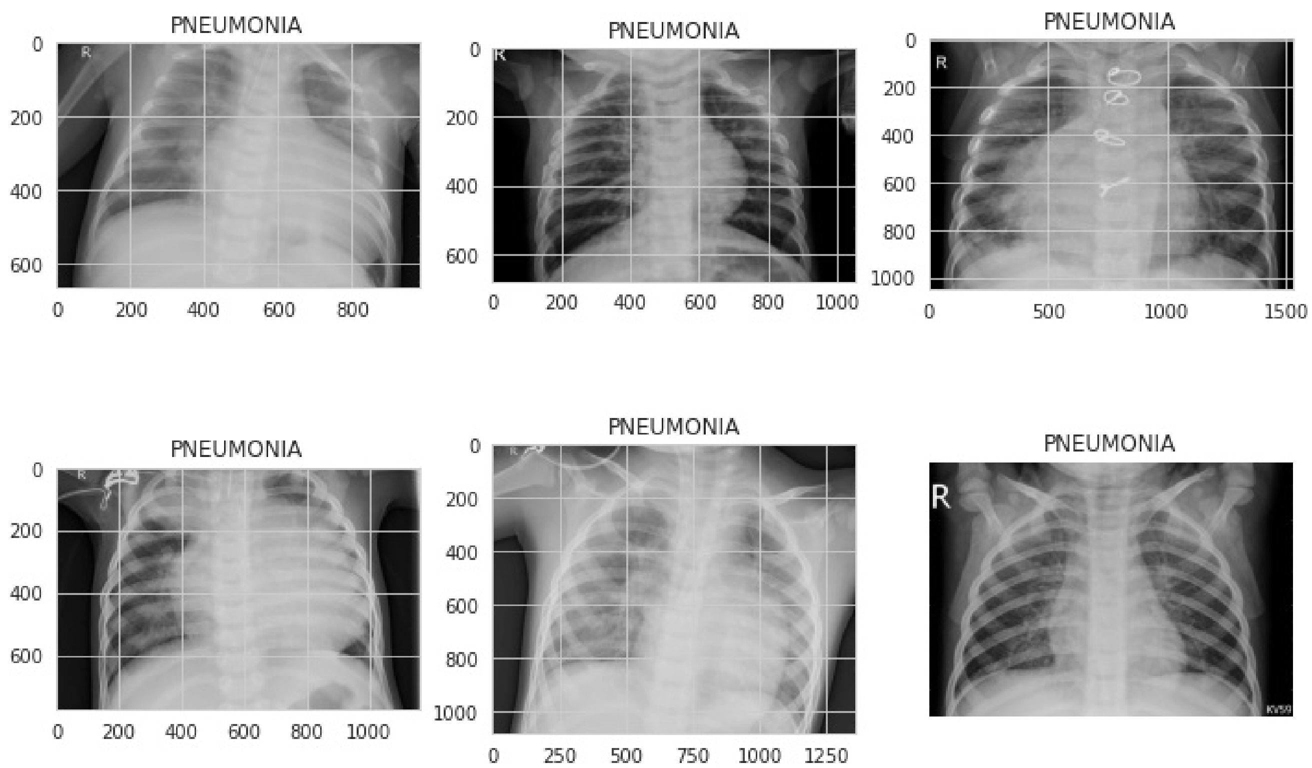
In [17]:

```python
def show_sample_images (number, normal=True):
    plt.figure(figsize=(12,12))
    if normal == True:
       for n in range(number):
          normal_img = os.listdir(train_n_path)[n]
          normal_img_address = train_n_path+normal_img
          normal_load = Image.open(normal_img_address)
          ax = plt.subplot(number/2,number/2,n+1)
          plt.imshow(normal_load, cmap ='gray')
          plt.title("NORMAL")
    else:
        for n in range(number):
            pneumonia_img = os.listdir(train_p_path)[n]
            pneumonia_img_address = train_p_path+pneumonia_img
            pneumonia_load = Image.open(pneumonia_img_address)
            ax = plt.subplot(number/2,number/2,n+1)
            plt.imshow(pneumonia_load, cmap ='gray')
            plt.title("PNEUMONIA")

    plt.axis("off")

# Show normal images
show_sample_images(6)
```



In [18]:

```
# show pneumonia images
show_sample_images(6,False)
```



Applying histogram equalization to improve images contrast (better visualisation). More about histogram equalization can be found at https://homepages.inf.ed.ac.uk/rbf/HIPR2/ htm and also documentation can be found here (how to implement it) here. As you can see below, this gives a better visualisation of the images. Histogram Equalization is a way of spreading out highly populated intensity values which could be causing the image to have a degraded quality.

https://towardsdatascience.com/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085

In [19]:

```
#!pip install scikit-image
from skimage import exposure
```

In [20]:

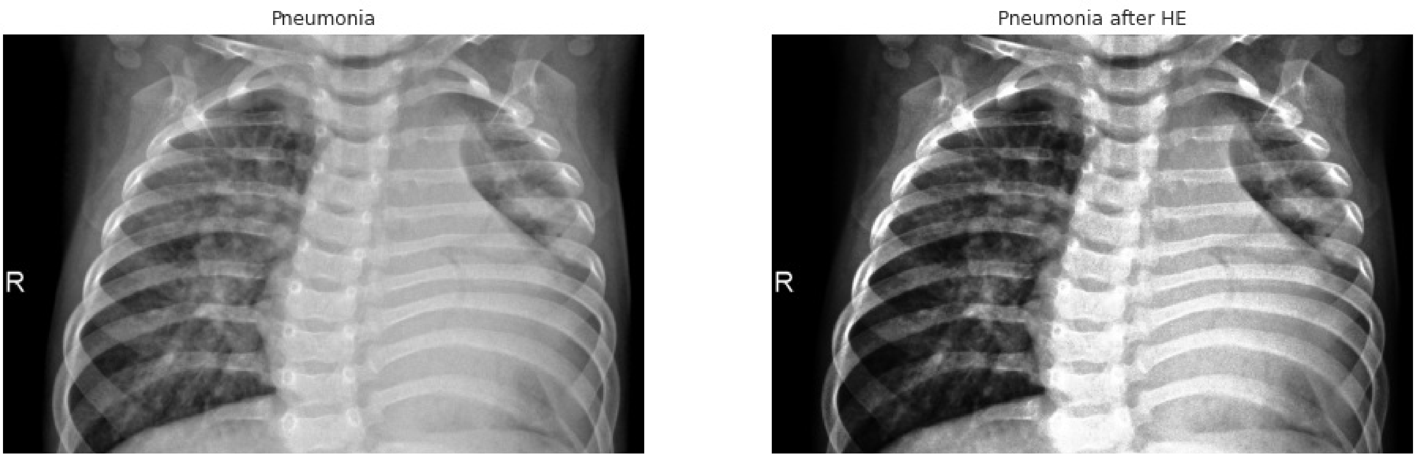
```python
def equlize_hist_ (image):
  _image = np.asarray(image)
  image_eq = exposure.equalize_hist(_image)
  return image_eq
```

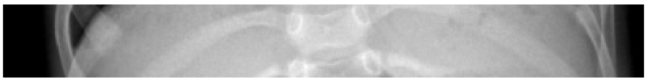
In [21]:

```python
def expose_imgae(Normal=True):
    if Normal==True:
        # Choose normal random image: generate random number between 1 and the number of normal images in the training set
        random_img_ind= np.random.randint(0,len(os.listdir(train_n_path)))
        # image file name
        img_expose_name = os.listdir(train_n_path)[random_img_ind]
        # path to the image
        img_expose_address = train_n_path+img_expose_name
        # load mage
        img_expose = Image.open(img_expose_address)
        img = np.asarray(img_expose)
        image_eq = equlize_hist_(img)
        figure1 = plt.figure(figsize= (16,16))
        img_1 = figure1.add_subplot(1,2,1)
        img_plot = plt.imshow(img, cmap = 'gray')
        img_1.set_title('Normal')
        plt.axis("off")
        img2 = figure1.add_subplot(1, 2, 2)
        img_plot = plt.imshow(image_eq, cmap = 'gray')
        img2.set_title('Normal after HE')
        plt.axis("off")
    else:
        # Choose normal random image: generate random number between 1 and the number of normal images in the training set
        random_img_ind= np.random.randint(0,len(os.listdir(train_p_path)))
        # image file name
        img_expose_name = os.listdir(train_p_path)[random_img_ind]
        # path to the image
        img_expose_address = train_p_path+img_expose_name
        # load mage
        img_expose = Image.open(img_expose_address)
        img = np.asarray(img_expose)
        image_eq = equlize_hist_(img)
        figure1 = plt.figure(figsize= (16,16))
        img_1 = figure1.add_subplot(1,2,1)
        img_plot = plt.imshow(img, cmap = 'gray')
        img_1.set_title('Pneumonia')
        plt.axis("off")
        img2 = figure1.add_subplot(1, 2, 2)
        img_plot = plt.imshow(image_eq, cmap = 'gray')
        img2.set_title('Pneumonia after HE')
        plt.axis("off")
```

In [22]:

```python
# expose_image() will show random normal image
expose_imgae(False)
```





As shown in the image above it is clear to see why exposing an image is very import, it adds more definition to the photo and makes it a lot clearer as well as increasing the contrast. This will allow the model to make predictions with a much higher accuracy than if the images were not exposed.

https://www.ijert.org/trying-to-see-low-exposure-images-using-cnn

Build CNN Model

First step, we want to arrange the data in different constructs (x_train, y_train, x_test, y_test, x_val,y_val), etc. . .):

It will also help to split the data into and 80/20 split of training and testing data.

In [23]:

```
x_train = []
y_train = []
x_val = []
y_val = []

x_test = []
y_test = []

x=0
# This will put 80% of the data into the tarining set and the remaining 20 % into the val
idation set.
for feature, label in train:
  if (x % 5 == 0):
    x_val.append(feature)
    y_val.append(label)
  else:
    x_train.append(feature)
    y_train.append(label)
  x+=1

#for feature, label in train:
#    x_train.append(feature)
#    y_train.append(label)
for feature, label in test:
    x_test.append(feature)
    y_test.append(label)

#for feature, label in val:
#    x_val.append(feature)
#    y_val.append(label)
```

In [24]:

```
print(f'The number of images in the training set is {len(x_train)}')
print(f'The number of images in the training set is {len(x_val)}')
```

```
The number of images in the training set is 4172
The number of images in the training set is 1044
```

In [25]:

```
# Normalise the data
x_train = np.array(x_train) / 255.0
x_val = np.array(x_val) / 255.0
x_test = np.array(x_test) / 255.0
```

Resize the arrays for deep learning into the needed shape

In [26]:

```
# resize data for deep learning
x_train = x_train.reshape(-1, img_size, img_size, 3)
y_train = np.array(y_train)
x_val = x_val.reshape(-1, img_size, img_size, 3)
y_val = np.array(y_val)
x_test = x_test.reshape(-1, img_size, img_size, 3)
y_test = np.array(y_test)
```

In [27]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

Image Data Generator

Image Data Generator In this part, we will build a CNN model. First, we will make use of Kerass ImageDatGenerator to perform data augmentation (see lecture notes for more on data augmentation). Recall, data augmentation help improve the performance of the model by generating more data via applying a geometric transformation (e.g. translation, rotation, scaling, shearing, etc. . .) to existing data/ images.

This will allow for a more accurate response by the model as it will give the model more data to make its predictions on. https://analyticsindiamag.com/image-data-augmentation-impacts-performance-of-image-classification-with-codes/

In [28]:

```
# n_steps = int(math.ceil(1. * training_set.samples / batch_size))
# data generator
datagen = ImageDataGenerator(
        featurewise_center=False, # set input mean to 0 over the dataset
        samplewise_center=False, # set each sample mean to 0
        featurewise_std_normalization=False, # divide inputs by std of the dataset
        samplewise_std_normalization=False, # divide each input by its std
        zca_whitening=False, # apply ZCA whitening
        rotation_range = 30, # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.1, # randomly shift images horizontally (fraction of total wid
th)
        height_shift_range=0.1, # randomly shift images vertically (fraction of total heig
ht)
        horizontal_flip = True, # randomly flip images
        vertical_flip=False) # randomly flip images
```

In [29]:

```
datagen.fit(x_train)
```

2 Baseline Model

The baseline model which has been implemented here has been taken from the paper below . Their research was also on pnumeonia image classification which allowed the baseline model to be used here for our own image classification. The same model has been implemented here to help with a baseline model for classifying what images appear to have pneomnia. It uses the activation relu throughout until the very end when it uses rmsprop. After the model has been created the summary of the model has been displayed.

https://ieeexplore.ieee.org/document/9057809/figures#figures

In [32]:

```
model = Sequential() #Creates the model as sequential

model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same'  ,
input_shape = (img_size,img_size,3))) #Gives the input chnage of the images


model.add(Activation (activation= 'relu')) #Adds an activation layer
```

```
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' ,
input_shape = (img_size,img_size,3)))

model.add(Activation (activation= 'relu'))

model.add(MaxPool2D((2,2) , strides = 3)) #Adds a MaxPool2d

model.add(Dropout(0.2)) #Applies a dropout to the model

model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' ,
input_shape = (img_size,img_size,3))) #Adds a Conv2D layer

model.add(Activation (activation= 'relu'))

model.add(Conv2D(128 , (2,2) , strides = 1)) #Adds a Conv2D layer

model.add(Activation (activation= 'relu'))

model.add(MaxPool2D((2,2) , strides = 2)) #Applies a maxpool2d function

model.add(Dropout(0.5)) #does a drop out here

model.add(Flatten()) #Flattens the model

model.add(Dense(units = 256)) #Adds dense layers

model.add(Activation (activation= 'relu'))

model.add(Dropout(0.5)) ##Another drop out


model.add(Dense(units = 512))

model.add(Dropout(0.5)) #Last drop out

model.add(Activation (activation= 'relu'))

model.add(Dense(units = 1)) #Adds a dense layer

model.add(Activation (activation= 'sigmoid'))

model.compile(optimizer = "rmsprop" , loss = 'binary_crossentropy' , metrics = ['accurac
y']) #Optimozier set to Rmsprop

model.save_weights('model.h5')
```

In [33]:

```
# model summary (see the number of trainable parameters)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 90, 90, 32)        896       
                                                                 
 activation_7 (Activation)   (None, 90, 90, 32)        0         
                                                                 
 conv2d_5 (Conv2D)           (None, 90, 90, 32)        9248      
                                                                 
 activation_8 (Activation)   (None, 90, 90, 32)        0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 30, 30, 32)       0         
 2D)                                                             
                                                                 
 dropout_4 (Dropout)         (None, 30, 30, 32)        0         
                                                                 
 conv2d_6 (Conv2D)           (None, 30, 30, 64)        18496     
                                                                 
 activation_9 (Activation)   (None, 30, 30, 64)        0         
```

```
 conv2d_7 (Conv2D)           (None, 29, 29, 128)       32896

 activation_10 (Activation)  (None, 29, 29, 128)       0

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 dropout_5 (Dropout)         (None, 14, 14, 128)       0

 flatten_1 (Flatten)         (None, 25088)             0

 dense_3 (Dense)             (None, 256)               6422784

 activation_11 (Activation)  (None, 256)               0

 dropout_6 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 512)               131584

 dropout_7 (Dropout)         (None, 512)               0

 activation_12 (Activation)  (None, 512)               0

 dense_5 (Dense)             (None, 1)                 513

 activation_13 (Activation)  (None, 1)                 0

=================================================================
Total params: 6,616,417
Trainable params: 6,616,417
Non-trainable params: 0
_________________________________________________________________
```

- **Conv2d - This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. - https://keras.io/api/layers/convolution_layers/convolution2d/**
- **MaxPooling2D - Takes the input and down samples it by its height and width given the input window. - https://keras.io/api/layers/pooling_layers/max_pooling2d/**
- **Dropout layer - Is used to help reduce the amount of overfitting occuring by droping weights at a probablility. - https://keras.io/api/layers/regularization_layers/dropout/**
- **Flatten - Takes a multidimentional output and chnages it to being linear. - https://datascience.stackexchange.com/questions/44124/when-to-use-dense-conv1-2d-dropout-flatten-and-all-the-other-layers#:~:text=Dropout%20and%20Flatten,weight%20updation%20for%20the%20edge.**
- **Dense - This is the most common layer added to a model to connect other layers within the model. - https://www.tutorialspoint.com/keras/keras_dense_layer.htm**

In [ ]:

```
total = len(os.listdir(train_n_path)) + len(os.listdir(train_p_path))
neg = len(os.listdir(train_n_path))
pos = len(os.listdir(train_p_path))
weight_for_0 = 1 /neg * (total/2.0)
weight_for_1 = 1/pos *(total/2.0)
class_weight = {0: weight_for_0, 1: weight_for_1}
class_weight
```

Out[ ]:

```
{0: 1.9448173005219984, 1: 0.6730322580645162}
```

After initally running the model with an epoch of twenty-five, it was clear to see that after around the 13th/14th epoch the model does not increase the accuracy but remians the same along with the val accuracy which does not chnage after this point. Due to this the epoch has been lowered to seventeen and the model rerun with the new epoch to reduce the chances of overfitting occuring with the higher epoch but still lowering the learning rate to reduce when it beghins to plateu to get the most accurate response. Please see image below for model

rate to reduce when it begins to plateu to get the most accurate response. Please see image below for model with epoch = 25

```
Epoch 8/25
261/261 [============================] - 10s 40ms/step - loss: 0.2739 - accuracy: 0.8864 - val_loss: 0.2556 - val_accuracy: 0.8898 - lr: 2.0000e-04
Epoch 9/25
261/261 [============================] - ETA: 0s - loss: 0.2691 - accuracy: 0.8914
Epoch 9: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
261/261 [============================] - 10s 40ms/step - loss: 0.2691 - accuracy: 0.8914 - val_loss: 0.2547 - val_accuracy: 0.8946 - lr: 2.0000e-04
Epoch 10/25
261/261 [============================] - 14s 53ms/step - loss: 0.2283 - accuracy: 0.9027 - val_loss: 0.1905 - val_accuracy: 0.9301 - lr: 4.0000e-05
Epoch 11/25
261/261 [============================] - 10s 40ms/step - loss: 0.2384 - accuracy: 0.9056 - val_loss: 0.1858 - val_accuracy: 0.9195 - lr: 4.0000e-05
Epoch 12/25
261/261 [============================] - ETA: 0s - loss: 0.2245 - accuracy: 0.9060
Epoch 12: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
261/261 [============================] - 10s 39ms/step - loss: 0.2245 - accuracy: 0.9060 - val_loss: 0.1929 - val_accuracy: 0.9253 - lr: 4.0000e-05
Epoch 13/25
261/261 [============================] - 11s 42ms/step - loss: 0.2163 - accuracy: 0.9120 - val_loss: 0.2087 - val_accuracy: 0.9119 - lr: 8.0000e-06
Epoch 14/25
261/261 [============================] - ETA: 0s - loss: 0.2264 - accuracy: 0.9051
Epoch 14: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
261/261 [============================] - 10s 40ms/step - loss: 0.2264 - accuracy: 0.9051 - val_loss: 0.2272 - val_accuracy: 0.9061 - lr: 8.0000e-06
Epoch 15/25
261/261 [============================] - 10s 40ms/step - loss: 0.2164 - accuracy: 0.9082 - val_loss: 0.2224 - val_accuracy: 0.9128 - lr: 1.6000e-06
Epoch 16/25
261/261 [============================] - ETA: 0s - loss: 0.2110 - accuracy: 0.9130
Epoch 16: ReduceLROnPlateau reducing learning rate to 1e-06.
261/261 [============================] - 10s 40ms/step - loss: 0.2110 - accuracy: 0.9130 - val_loss: 0.2038 - val_accuracy: 0.9138 - lr: 1.6000e-06
Epoch 17/25
261/261 [============================] - 11s 42ms/step - loss: 0.2147 - accuracy: 0.9123 - val_loss: 0.1901 - val_accuracy: 0.9282 - lr: 1.0000e-06
Epoch 18/25
261/261 [============================] - 11s 41ms/step - loss: 0.2126 - accuracy: 0.9108 - val_loss: 0.2042 - val_accuracy: 0.9157 - lr: 1.0000e-06
Epoch 19/25
261/261 [============================] - 10s 39ms/step - loss: 0.2087 - accuracy: 0.9161 - val_loss: 0.2032 - val_accuracy: 0.9243 - lr: 1.0000e-06
Epoch 20/25
261/261 [============================] - 10s 40ms/step - loss: 0.2147 - accuracy: 0.9161 - val_loss: 0.2165 - val_accuracy: 0.9167 - lr: 1.0000e-06
Epoch 21/25
261/261 [============================] - 10s 40ms/step - loss: 0.2189 - accuracy: 0.9135 - val_loss: 0.1999 - val_accuracy: 0.9234 - lr: 1.0000e-06
Epoch 22/25
261/261 [============================] - 10s 40ms/step - loss: 0.2153 - accuracy: 0.9123 - val_loss: 0.2000 - val_accuracy: 0.9234 - lr: 1.0000e-06
Epoch 23/25
261/261 [============================] - 10s 40ms/step - loss: 0.2197 - accuracy: 0.9120 - val_loss: 0.2094 - val_accuracy: 0.9157 - lr: 1.0000e-06
Epoch 24/25
261/261 [============================] - 10s 40ms/step - loss: 0.2160 - accuracy: 0.9094 - val_loss: 0.2062 - val_accuracy: 0.9176 - lr: 1.0000e-06
```

The batch size has been set to being 18. This is due to reasearch into what the optimal batch size is for training images. Any higher of a batch size can't start to overfit and produced untrue results along with not enough computing memory available to implement a higher batch size to be used.

https://www.sciencedirect.com/science/article/pii/S2405959519303455#:~:text=In%20practical%20terms%2C%20

In [ ]:

```
batch_size = 18
n_epochs = 17
```

To ensure that the learning rate is always set to the most optimum value I have used the Keras function ReduceLROnPlateau. This will allow the learning rate to reduce by a factor of 0.2 once the loss begins to plateu. This allows for the minimum loss surface to be found more quickly as the jumps between the required answer will be less spread out adn reduce the loss.

http://www.bdhammel.com/learning-rates/

In [ ]:

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
        patience=2, verbose=1, factor=0.2, min_lr=0.000001)
history = model.fit(datagen.flow(x_train, y_train,
                    batch_size=batch_size), epochs=n_epochs,
                    validation_data=datagen.flow(x_val, y_val),
                    callbacks=[learning_rate_reduction],
                    class_weight=class_weight)
```

```
Epoch 1/17
232/232 [==============================] - 21s 42ms/step - loss: 0.6913 - accuracy: 0.579
6 - val_loss: 0.7465 - val_accuracy: 0.6389 - lr: 0.0010
Epoch 2/17
232/232 [==============================] - 9s 40ms/step - loss: 0.5894 - accuracy: 0.7028
- val_loss: 0.7079 - val_accuracy: 0.6887 - lr: 0.0010
Epoch 3/17
232/232 [==============================] - 9s 40ms/step - loss: 0.5181 - accuracy: 0.7383
- val_loss: 0.5537 - val_accuracy: 0.6992 - lr: 0.0010
```

```
Epoch 4/17
232/232 [==============================] - 9s 40ms/step - loss: 0.4476 - accuracy: 0.7898
- val_loss: 0.3227 - val_accuracy: 0.8410 - lr: 0.0010
Epoch 5/17
232/232 [==============================] - 9s 40ms/step - loss: 0.4158 - accuracy: 0.8039
- val_loss: 0.2946 - val_accuracy: 0.8831 - lr: 0.0010
Epoch 6/17
232/232 [==============================] - 9s 40ms/step - loss: 0.4101 - accuracy: 0.8281
- val_loss: 0.2616 - val_accuracy: 0.8898 - lr: 0.0010
Epoch 7/17
232/232 [==============================] - 9s 40ms/step - loss: 0.3788 - accuracy: 0.8267
- val_loss: 0.3940 - val_accuracy: 0.8161 - lr: 0.0010
Epoch 8/17
231/232 [============================>.] - ETA: 0s - loss: 0.3592 - accuracy: 0.8339
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
232/232 [==============================] - 9s 40ms/step - loss: 0.3591 - accuracy: 0.8344
- val_loss: 0.2743 - val_accuracy: 0.8764 - lr: 0.0010
Epoch 9/17
232/232 [==============================] - 9s 40ms/step - loss: 0.2489 - accuracy: 0.8967
- val_loss: 0.3224 - val_accuracy: 0.8458 - lr: 2.0000e-04
Epoch 10/17
232/232 [==============================] - 9s 40ms/step - loss: 0.2484 - accuracy: 0.8938
- val_loss: 0.1788 - val_accuracy: 0.9330 - lr: 2.0000e-04
Epoch 11/17
232/232 [==============================] - 9s 40ms/step - loss: 0.2519 - accuracy: 0.8977
- val_loss: 0.2105 - val_accuracy: 0.9148 - lr: 2.0000e-04
Epoch 12/17
231/232 [============================>.] - ETA: 0s - loss: 0.2438 - accuracy: 0.9013
Epoch 12: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
232/232 [==============================] - 9s 40ms/step - loss: 0.2455 - accuracy: 0.9010
- val_loss: 0.2399 - val_accuracy: 0.8956 - lr: 2.0000e-04
Epoch 13/17
232/232 [==============================] - 9s 40ms/step - loss: 0.2190 - accuracy: 0.9065
- val_loss: 0.1773 - val_accuracy: 0.9205 - lr: 4.0000e-05
Epoch 14/17
231/232 [============================>.] - ETA: 0s - loss: 0.2116 - accuracy: 0.9121
Epoch 14: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
232/232 [==============================] - 9s 40ms/step - loss: 0.2121 - accuracy: 0.9123
- val_loss: 0.2055 - val_accuracy: 0.9167 - lr: 4.0000e-05
Epoch 15/17
232/232 [==============================] - 9s 40ms/step - loss: 0.2228 - accuracy: 0.9116
- val_loss: 0.2022 - val_accuracy: 0.9224 - lr: 8.0000e-06
Epoch 16/17
231/232 [============================>.] - ETA: 0s - loss: 0.2263 - accuracy: 0.9073
Epoch 16: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
232/232 [==============================] - 9s 40ms/step - loss: 0.2259 - accuracy: 0.9075
- val_loss: 0.1907 - val_accuracy: 0.9224 - lr: 8.0000e-06
Epoch 17/17
232/232 [==============================] - 9s 40ms/step - loss: 0.2270 - accuracy: 0.9099
- val_loss: 0.1813 - val_accuracy: 0.9291 - lr: 1.6000e-06
```

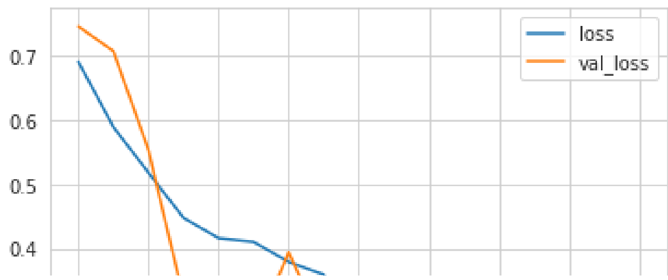
Analyse Results

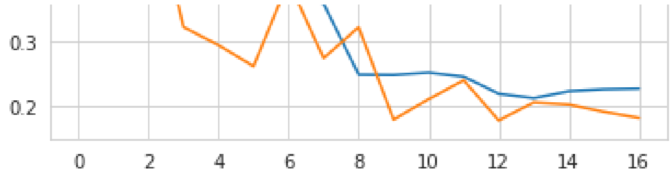
In [ ]:

```
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc73e31d650>
```



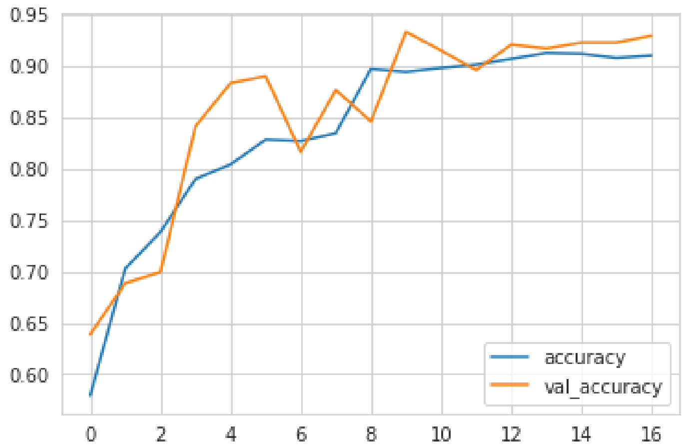


In [ ]:

```
losses[['accuracy','val_accuracy']].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc73e1f5710>
```



Check the testing accuracy/ loss

In [ ]:

```
print("Loss of the model is - " , model.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")
```

```
20/20 [==============================] - 0s 14ms/step - loss: 0.4364 - accuracy: 0.8734
Loss of the model is -  0.43635794520378113
20/20 [==============================] - 0s 7ms/step - loss: 0.4364 - accuracy: 0.8734
Accuracy of the model is -  87.33974099159241 %
```

The baseline model which I have implemented here has an accuracy of just over 86% with a loss of just over 0.45. This accuracy is quite high given the restrictions of memory and capacity running on the machine. This shows that 86% of the time the model will be able to correctly identify if a patient either has or doesn't have pneumonia. With the loss of 0.43, it also shows that the model is not getting a lot of loss in the model which could reduce the effect of using this paticular model.

Save the model and make some predictions:

In [ ]:

```
from tensorflow.keras.models import load_model
model.save('CNN_PNEUMONIA.h5')
#cnn_model = load_model("./CNN_PNEUMONIA.h5")

predictions = (model.predict(x_test)>0.5).astype("int32")
predictions = predictions.reshape(1,-1)[0]
predictions[:15]
```

Out[ ]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [ ]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test, predictions, target_names = ['Pneumonia(Class 0)','N
ormal (Class 1)']))
```

```
                     precision    recall  f1-score   support

Pneumonia(Class 0)       0.84      0.82      0.83       234
  Normal (Class 1)       0.89      0.91      0.90       390

        accuracy                           0.87       624
       macro avg       0.87      0.86      0.86       624
    weighted avg       0.87      0.87      0.87       624
```

The precision of the model is 0.84 for Pneumonia and 0.89 for Normal. This means that for x rays which the model classified as having Pneumonia, 84% of the time the model was correct in predicting this. As for Normal (no Pneumonia) the model was more accurate at predicting that they didn't have it with an acuuracy of 89%.

The recall of the model is how well the model managed to find all postive instances. For Pneumonia, the model managed to correctly classify 82% of the overall Pneumonia cases. For the Normal images the model did very well in identifying all of the inastces of the pateint not having Pneumonia with an accuracy of 91%.

https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html

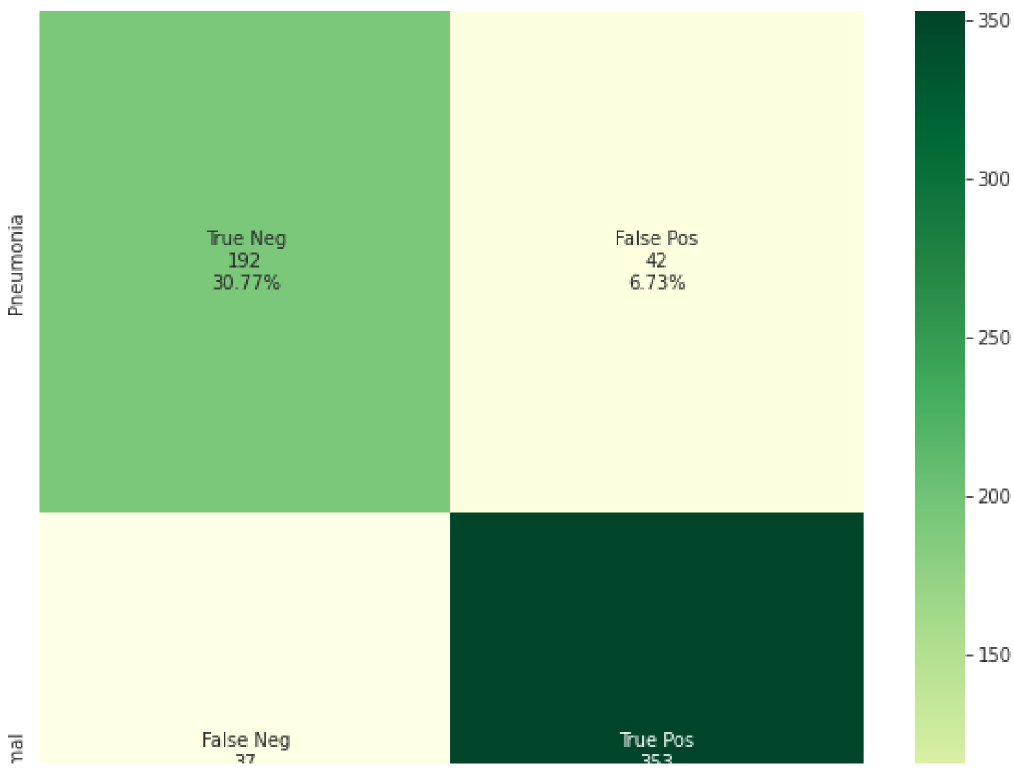
Plot confusion matrix

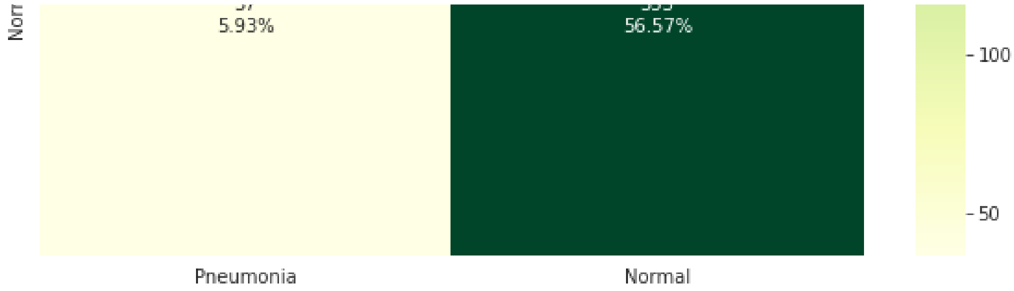
In [ ]:

```
cf_matrix = confusion_matrix(y_test, predictions)
plt.figure(figsize = (10,10))
classes = ['Pneumonia','Normal']
labels = ['TN','FP','FN','TP']
labels = np.asarray(labels).reshape(2,2)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
  cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
  cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
  zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap= "YlGn" ,
  xticklabels = classes,yticklabels = classes
  )
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc73e08b790>
```





From the matrix above you can see the percentage of what the spread of the models' predictions are. It appears to be that the model does very well in predicting when the patient does in fact have pneumonia this is important as this is the most essential reading to get back from the model so is good that this is a high percentage at 56%. The second best spread of results was the true negative result with 31%. Both of these results are good as the model is identifying correctly. Finally, both false positives and false negatives are both relatively low percentage with around 7% of what the model predicting a false postive, with the lowest being a false negative at 6%. Again, this is very encouraging as the worst outcome of the models' predictions would be saying a patient doesn't have pneumonia when they do which only occurs in 6% of the time.

In [ ]:

```
# store actual class labels and predicted ones in a dataframe
results = pd.DataFrame({'Actual':y_test,'Predicted':predictions})
incorrect_df = results[results.Actual!=results.Predicted]
incorrect_df.head()
```

Out[ ]:

	Actual	Predicted
34	0	1
49	0	1
57	0	1
77	0	1
91	0	1

In [ ]:

```
# manual calculation of of results
print(f'Accuracy is {round((results.shape[0]-incorrect_df.shape[0])/results.shape[0],2)*1
00} %')
```

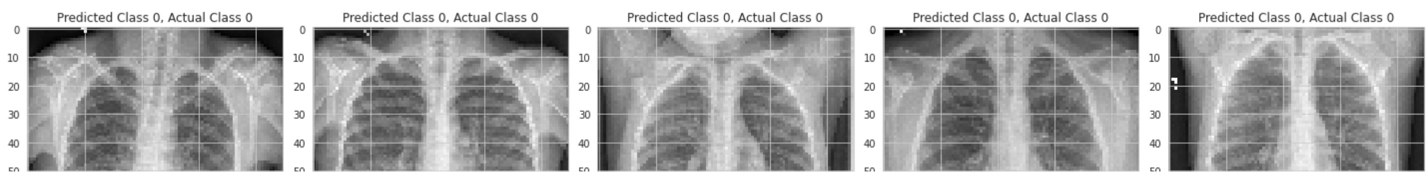
```
Accuracy is 87.0 %
```

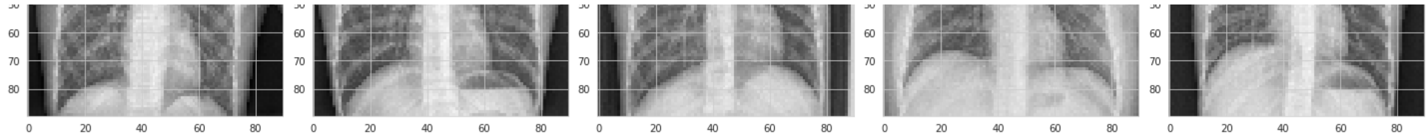
In [ ]:

```
#show some examples
correct = np.nonzero(predictions == y_test)[0]
incorrect = np.nonzero(predictions != y_test)[0]
```

In [ ]:

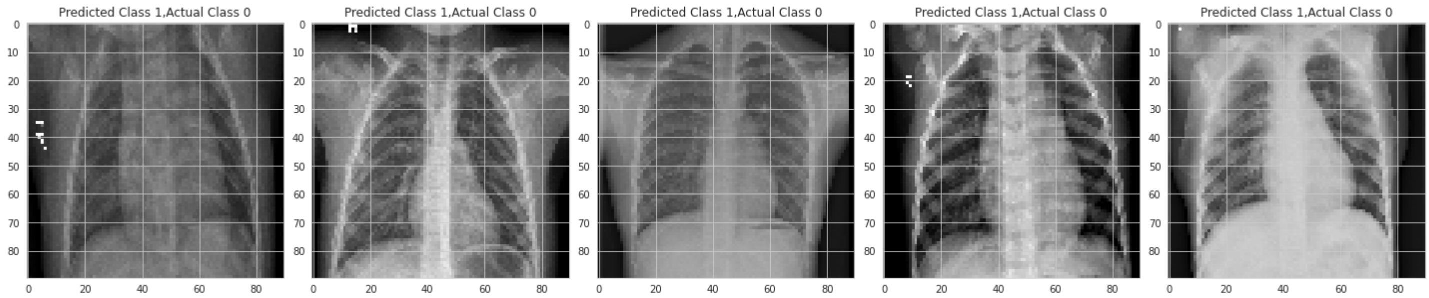
```
i = 0
figure = plt.figure(figsize= (20,20))
for c in correct[:5]:
  ax = plt.subplot(5,5,i+1)
  plt.imshow(x_test[c].reshape(img_size,img_size,3), cmap="gray", interpolation='none')
  plt.title("Predicted Class {}, Actual Class {}".format(predictions[c], y_test[c]))
  plt.tight_layout()
  i += 1
```





In [ ]:

```
i = 0
figure = plt.figure(figsize= (20,20))
for c in incorrect[:5]:
  ax = plt.subplot(5,5,i+1)
  plt.imshow(x_test[c].reshape(img_size,img_size,3), cmap="gray", interpolation='none')
  plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
  plt.tight_layout()
  i += 1
```



3 Transfer Learning

Pre-trained models

- **VGG19**
- **InceptionResntnetV2**

VGG19

VGG19 is a convolutional neural network which has 19 layers with 16 being convolutional and the remaning 3 layers fully connected. This is one of the most popular image classifications methods used due to the 3x3 filters it uses in each convolutional layer. There are previous studies which have been done into using multiple different transfer learning models to detect pneumonia and VGG19 returned the best accuracy in the experiment with 87% accurate so has been used in this study.

https://link.springer.com/article/10.1007/s12652-021-03488-z#:~:text=VGG19%20is%20trained%20on%20the,filters%20in%20each%20convolutional%20layer.

https://www.sciencedirect.com/science/article/pii/S0167865520304414?casa_token=U5gzfte-2LsAAAAA:aG2ReAL_g1gR8pz519osGYworanJqvNZKQlpeV8z_s4y8KeKp-N6mvRV49yfG1vFAhEpb2PiIV0

In [ ]:

```
from keras.applications.vgg19 import VGG19

# Notice 1st time this is being run, it will download the weights for the ResNet model
tf.keras.backend.clear_session()
base_model = VGG19(
  weights='imagenet',
  input_shape=(img_size, img_size, 3),
  include_top=False)
# freeze the layers
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/
vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 2s 0us/step
80150528/80134624 [==============================] - 2s 0us/step
```

In [ ]:

```
def get_pretrained():
#Input shape = [width, height, color channels]
  inputs = layers.Input(shape=(img_size,img_size, 3))
  x = base_model(inputs)
  # Head
  x = layers.GlobalAveragePooling2D()(x)
  x = layers.Dense(128, activation='relu')(x)
  x = layers.Dropout(0.1)(x)
  #Final Layer (Output)
  output = layers.Dense(1, activation='sigmoid')(x)
  model = tf.keras.Model(inputs=[inputs], outputs=output)
  return model
```

In [ ]:

```
model_pretrained = get_pretrained()
model_pretrained.compile(loss='binary_crossentropy', optimizer = tf.keras.optimizers.Ada
m(learning_rate=0.00005), metrics='binary_accuracy')
model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 90, 90, 3)]       0

 vgg19 (Functional)          (None, 2, 2, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               65664

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 20,090,177
Trainable params: 65,793
Non-trainable params: 20,024,384
_________________________________________________________________
```

In [ ]:

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_binary_accuracy',
        patience=2, verbose=1, factor=0.2, min_lr=0.000001)
```

In [ ]:

```
history_tl = model_pretrained.fit(datagen.flow(x_train,y_train, batch_size = batch_size)
,
epochs = n_epochs , validation_data = datagen.flow(x_val,y_val) ,
callbacks = [learning_rate_reduction],
steps_per_epoch = x_train.shape[0]/batch_size,
class_weight = class_weight
)
```

```
Epoch 1/17
231/231 [==============================] - 13s 50ms/step - loss: 0.6583 - binary_accuracy
: 0.6906 - val_loss: 0.5418 - val_binary_accuracy: 0.8190 - lr: 5.0000e-05
Epoch 2/17
231/231 [==============================] - 10s 42ms/step - loss: 0.5577 - binary_accuracy
: 0.7943 - val_loss: 0.5309 - val_binary_accuracy: 0.7989 - lr: 5.0000e-05
Epoch 3/17
231/231 [==============================] - 10s 43ms/step - loss: 0.4922 - binary_accuracy
: 0.8061 - val_loss: 0.4777 - val_binary_accuracy: 0.8266 - lr: 5.0000e-05
Epoch 4/17
231/231 [==============================] - 10s 43ms/step - loss: 0.4444 - binary_accuracy
: 0.8281 - val_loss: 0.4041 - val_binary_accuracy: 0.8611 - lr: 5.0000e-05
```

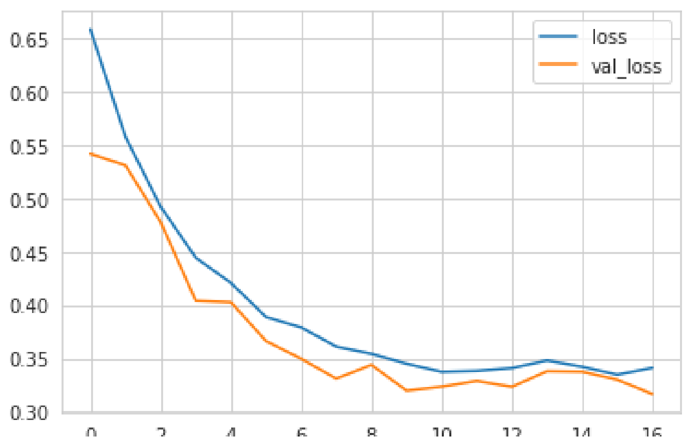
```
Epoch 5/17
231/231 [==============================] - 10s 43ms/step - loss: 0.4207 - binary_accuracy
: 0.8349 - val_loss: 0.4026 - val_binary_accuracy: 0.8496 - lr: 5.0000e-05
Epoch 6/17
231/231 [==============================] - 10s 43ms/step - loss: 0.3888 - binary_accuracy
: 0.8420 - val_loss: 0.3663 - val_binary_accuracy: 0.8678 - lr: 5.0000e-05
Epoch 7/17
231/231 [==============================] - 10s 43ms/step - loss: 0.3790 - binary_accuracy
: 0.8483 - val_loss: 0.3496 - val_binary_accuracy: 0.8697 - lr: 5.0000e-05
Epoch 8/17
231/231 [==============================] - 10s 44ms/step - loss: 0.3611 - binary_accuracy
: 0.8557 - val_loss: 0.3310 - val_binary_accuracy: 0.8630 - lr: 5.0000e-05
Epoch 9/17
231/231 [============================>.] - ETA: 0s - loss: 0.3537 - binary_accuracy: 0.85
53
Epoch 9: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
231/231 [==============================] - 10s 43ms/step - loss: 0.3543 - binary_accuracy
: 0.8547 - val_loss: 0.3439 - val_binary_accuracy: 0.8563 - lr: 5.0000e-05
Epoch 10/17
231/231 [==============================] - 10s 43ms/step - loss: 0.3450 - binary_accuracy
: 0.8564 - val_loss: 0.3197 - val_binary_accuracy: 0.8697 - lr: 1.0000e-05
Epoch 11/17
231/231 [============================>.] - ETA: 0s - loss: 0.3375 - binary_accuracy: 0.85
99
Epoch 11: ReduceLROnPlateau reducing learning rate to 1.9999999494757505e-06.
231/231 [==============================] - 10s 43ms/step - loss: 0.3374 - binary_accuracy
: 0.8598 - val_loss: 0.3234 - val_binary_accuracy: 0.8688 - lr: 1.0000e-05
Epoch 12/17
231/231 [==============================] - 10s 44ms/step - loss: 0.3383 - binary_accuracy
: 0.8598 - val_loss: 0.3289 - val_binary_accuracy: 0.8745 - lr: 2.0000e-06
Epoch 13/17
231/231 [==============================] - 10s 44ms/step - loss: 0.3409 - binary_accuracy
: 0.8588 - val_loss: 0.3233 - val_binary_accuracy: 0.8592 - lr: 2.0000e-06
Epoch 14/17
231/231 [============================>.] - ETA: 0s - loss: 0.3478 - binary_accuracy: 0.85
46
Epoch 14: ReduceLROnPlateau reducing learning rate to 1e-06.
231/231 [==============================] - 10s 43ms/step - loss: 0.3480 - binary_accuracy
: 0.8538 - val_loss: 0.3380 - val_binary_accuracy: 0.8669 - lr: 2.0000e-06
Epoch 15/17
231/231 [==============================] - 10s 45ms/step - loss: 0.3422 - binary_accuracy
: 0.8574 - val_loss: 0.3375 - val_binary_accuracy: 0.8640 - lr: 1.0000e-06
Epoch 16/17
231/231 [==============================] - 11s 45ms/step - loss: 0.3348 - binary_accuracy
: 0.8567 - val_loss: 0.3300 - val_binary_accuracy: 0.8736 - lr: 1.0000e-06
Epoch 17/17
231/231 [==============================] - 11s 45ms/step - loss: 0.3411 - binary_accuracy
: 0.8617 - val_loss: 0.3165 - val_binary_accuracy: 0.8803 - lr: 1.0000e-06
```

In [ ]:

```
losses = pd.DataFrame(model_pretrained.history.history)
losses[['loss','val_loss']].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc727aa4f90>
```

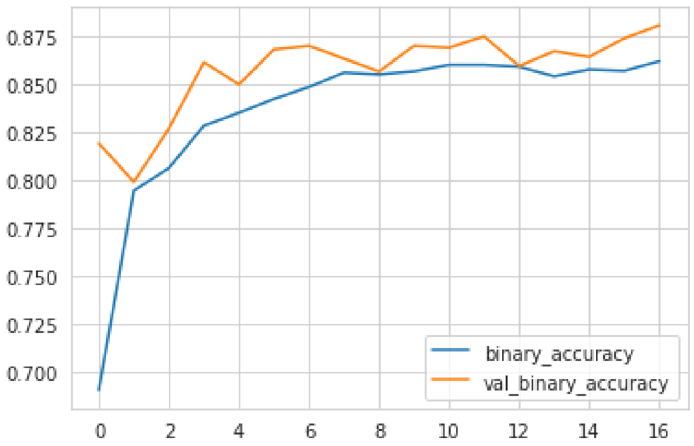


In [ ]:

```
losses[['binary_accuracy','val_binary_accuracy']].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc727a18790>
```



In [ ]:

```
print("Loss of the model is - " , model_pretrained.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model_pretrained.evaluate(x_test,y_test)[1]*100 , "%")
```

```
20/20 [==============================] - 1s 62ms/step - loss: 0.4292 - binary_accuracy: 0.8109
Loss of the model is -  0.42917367815971375
20/20 [==============================] - 1s 39ms/step - loss: 0.4292 - binary_accuracy: 0.8109
Accuracy of the model is -  81.08974099159241 %
```

VGG19 Result:

The outcome of using the transfer learning model VGG19 was that the accuracy was around 81% accurate. This is lower than the original baseline models accuracy and could be due to a number of factors. It could be that the model struggled to identify the tell tail signs of pneumonia and hasn't been pre trained well enough to identify the signs of pneumonia. It could also be that the epoch needed for this model needed to be higher but due to memory restraints this was not possible to be obtained.

The loss of the model is now at 0.43 which again is veryt promsing showing the model does not incure that high of loss.

https://thedatafrog.com/en/articles/image-recognition-transfer-learning/

In [ ]:

```
predictions = model_pretrained.predict(x_test)
pred_labels= np.where(predictions>0.5, 1, 0)
```

In [ ]:

```

```

In [ ]:

```
from tensorflow.keras.models import load_model
model.save('VGG19.h5')
#cnn_model = load_model("./CNN_PNEUMONIA.h5")

predictions = (model.predict(x_test)>0.5).astype("int32")
predictions = predictions.reshape(1,-1)[0]
predictions[:15]
```

Out[ ]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [ ]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test, predictions, target_names = ['Pneumonia(Class 0)','N
ormal (Class 1)']))
```

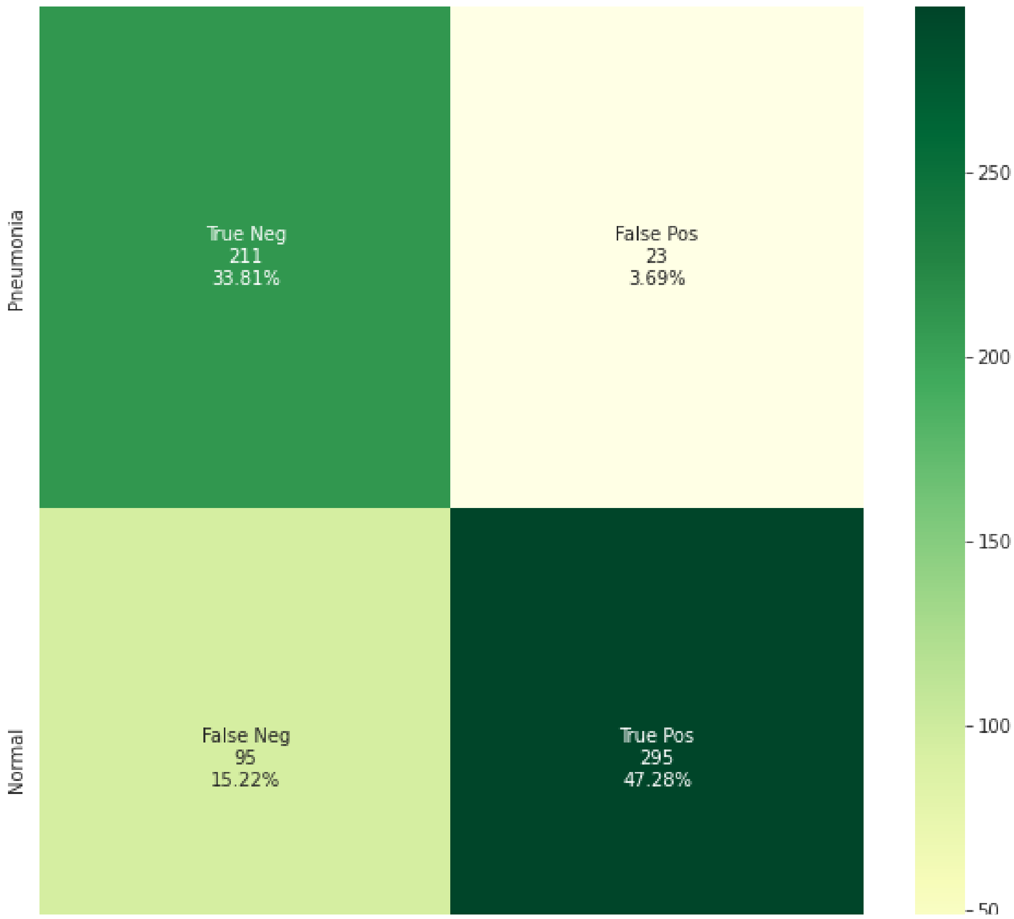
```
                    precision    recall  f1-score   support

Pneumonia(Class 0)       0.84      0.82      0.83       234
  Normal (Class 1)       0.89      0.91      0.90       390

          accuracy                           0.87       624
         macro avg       0.87      0.86      0.86       624
      weighted avg       0.87      0.87      0.87       624
```

In [ ]:

```
cf_matrix = confusion_matrix(y_test, pred_labels)
plt.figure(figsize = (10,10))
labels = ['TN','FP','FN','TP']
labels = np.asarray(labels).reshape(2,2)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap= "YlGn",
xticklabels = classes,yticklabels = classes)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7271f94d0>
```



It is clear where the model has gone wrong from the matrix above showing the spread of what the model identified. The model has a 15% likelihood of not diagnosing someone who has pneumonia by a returning a false negative. This is an issue has this is the worst outcome as this could impact people who would then believe they are not ill but in fact have pneumonia and could lead to health problems in the near future for them. The other precicitions are quite good on the other hand and the model is only let down by the false negatives.

In [ ]:

```
cf_matrix
```

Out[ ]:

```
array([[211,  23],
       [ 95, 295]])
```

Fine-tuning VGG19 Model

Notice that when we used pre-trained models, we haven't changed the weights, or retrained any of the ResNet layers. Simply put, we have updated the input and output of the models, while keeping all layers of the pre-trained models frozen. Although this is useful, in some cases you need to train some of the layers (update the weights of the model), and this is what is called finetuning the models. Here, we are going to unfreeze some layers and retrain. Note also, that we often keep lower layers frozen, because these capture generic features that may be shared with most images.

In [ ]:

```
#Fine tunning
base_model.trainable = True
# Retrain the last 10 layers (all lower layers will be kept frozen)
for layer in base_model.layers[:-10]:
    layer.trainable = False
```

This will allow the last 10 layers of the model to be retraind and unfrozen, this should allow for a more accurate result as the model will be able to be trained and have varying weights to try and get a more accurate result.

https://medium.com/@timsennett/unfreezing-the-layers-you-want-to-fine-tune-using-transfer-learning-1bad8cb72e5d

In [ ]:

```
model_pretrained.compile(loss='binary_crossentropy', optimizer = tf.keras.optimizers.Ada
m(learning_rate=0.000002), metrics='binary_accuracy')
model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 90, 90, 3)]       0         
                                                                 
 vgg19 (Functional)          (None, 2, 2, 512)         20024384  
                                                                 
 global_average_pooling2d (G  (None, 512)              0         
 lobalAveragePooling2D)                                          
                                                                 
 dense (Dense)               (None, 128)               65664     
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
```

```
_________________________________________________________________
Total params: 20,090,177
Trainable params: 17,764,609
Non-trainable params: 2,325,568
_________________________________________________________________
```

In [ ]:

```
history_ft = model_pretrained.fit(datagen.flow(x_train,y_train, batch_size = batch_size)
,
epochs = n_epochs , validation_data = datagen.flow(x_val, y_val) ,
callbacks = [learning_rate_reduction],
steps_per_epoch = x_train.shape[0]/batch_size,
class_weight = class_weight
)
```

```
Epoch 1/17
231/231 [==============================] - 17s 69ms/step - loss: 0.2585 - binary_accuracy
: 0.8921 - val_loss: 0.2294 - val_binary_accuracy: 0.9090 - lr: 2.0000e-06
Epoch 2/17
231/231 [==============================] - 15s 67ms/step - loss: 0.2012 - binary_accuracy
: 0.9180 - val_loss: 0.1657 - val_binary_accuracy: 0.9387 - lr: 2.0000e-06
Epoch 3/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1721 - binary_accuracy
: 0.9279 - val_loss: 0.1590 - val_binary_accuracy: 0.9358 - lr: 2.0000e-06
Epoch 4/17
232/231 [==============================] - ETA: 0s - loss: 0.1581 - binary_accuracy: 0.93
58
Epoch 4: ReduceLROnPlateau reducing learning rate to 1e-06.
231/231 [==============================] - 15s 66ms/step - loss: 0.1581 - binary_accuracy
: 0.9358 - val_loss: 0.1494 - val_binary_accuracy: 0.9377 - lr: 2.0000e-06
Epoch 5/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1477 - binary_accuracy
: 0.9410 - val_loss: 0.1484 - val_binary_accuracy: 0.9454 - lr: 1.0000e-06
Epoch 6/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1446 - binary_accuracy
: 0.9418 - val_loss: 0.1387 - val_binary_accuracy: 0.9444 - lr: 1.0000e-06
Epoch 7/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1355 - binary_accuracy
: 0.9473 - val_loss: 0.1629 - val_binary_accuracy: 0.9454 - lr: 1.0000e-06
Epoch 8/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1323 - binary_accuracy
: 0.9449 - val_loss: 0.1256 - val_binary_accuracy: 0.9521 - lr: 1.0000e-06
Epoch 9/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1297 - binary_accuracy
: 0.9504 - val_loss: 0.1093 - val_binary_accuracy: 0.9579 - lr: 1.0000e-06
Epoch 10/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1287 - binary_accuracy
: 0.9480 - val_loss: 0.1191 - val_binary_accuracy: 0.9569 - lr: 1.0000e-06
Epoch 11/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1244 - binary_accuracy
: 0.9489 - val_loss: 0.1451 - val_binary_accuracy: 0.9397 - lr: 1.0000e-06
Epoch 12/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1143 - binary_accuracy
: 0.9545 - val_loss: 0.1231 - val_binary_accuracy: 0.9511 - lr: 1.0000e-06
Epoch 13/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1124 - binary_accuracy
: 0.9540 - val_loss: 0.1280 - val_binary_accuracy: 0.9607 - lr: 1.0000e-06
Epoch 14/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1093 - binary_accuracy
: 0.9571 - val_loss: 0.1128 - val_binary_accuracy: 0.9511 - lr: 1.0000e-06
Epoch 15/17
231/231 [==============================] - 15s 66ms/step - loss: 0.1067 - binary_accuracy
: 0.9576 - val_loss: 0.1122 - val_binary_accuracy: 0.9540 - lr: 1.0000e-06
Epoch 16/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1093 - binary_accuracy
: 0.9533 - val_loss: 0.1229 - val_binary_accuracy: 0.9540 - lr: 1.0000e-06
Epoch 17/17
231/231 [==============================] - 15s 67ms/step - loss: 0.1044 - binary_accuracy
: 0.9590 - val_loss: 0.1017 - val_binary_accuracy: 0.9636 - lr: 1.0000e-06
```

In [ ]:

```
print("Loss of the model is - " , model_pretrained.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model_pretrained.evaluate(x_test,y_test)[1]*100 ,
"%")
```

```
20/20 [==============================] - 1s 41ms/step - loss: 0.1962 - binary_accuracy: 0
.9279
Loss of the model is -  0.19624868035316467
20/20 [==============================] - 1s 38ms/step - loss: 0.1962 - binary_accuracy: 0
.9279
Accuracy of the model is -  92.78846383094788 %
```

Evaluate and compare results of VGG19 against the baseline model

In [ ]:

```
predictions = model_pretrained.predict(x_test)
pred_labels= np.where(predictions>0.5, 1, 0)
```

In [ ]:

```
from tensorflow.keras.models import load_model
model.save('InceptionResnetV2.h5')
#cnn_model = load_model("./CNN_PNEUMONIA.h5")

predictions = (model.predict(x_test)>0.5).astype("int32")
predictions = predictions.reshape(1,-1)[0]
predictions[:15]
```

Out[ ]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [ ]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test, predictions, target_names = ['Pneumonia(Class 0)','N
ormal (Class 1)']))
```

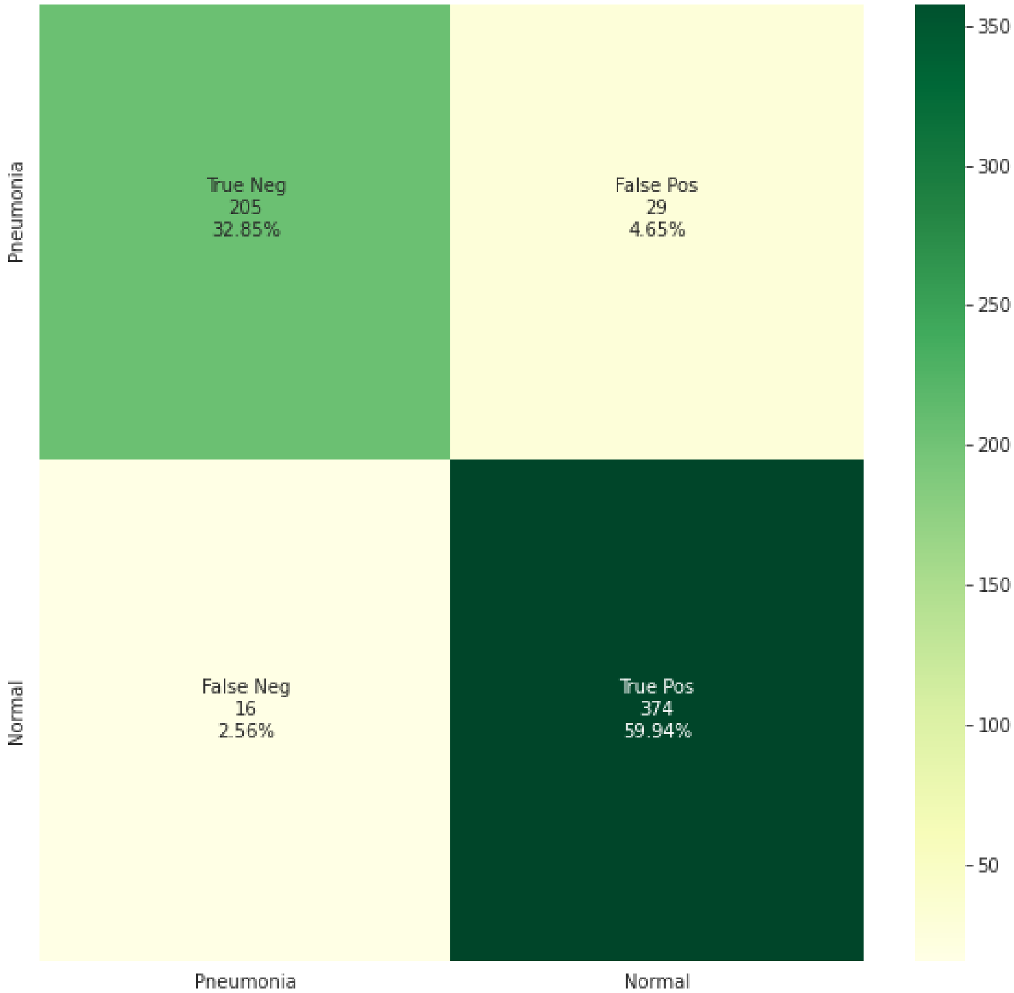
```
                    precision    recall  f1-score   support

Pneumonia(Class 0)       0.84      0.82      0.83       234
  Normal (Class 1)       0.89      0.91      0.90       390

          accuracy                           0.87       624
         macro avg       0.87      0.86      0.86       624
      weighted avg       0.87      0.87      0.87       624
```

In [ ]:

```
cf_matrix = confusion_matrix(y_test, pred_labels)
plt.figure(figsize = (10,10))
labels = ['TN','FP','FN','TP']
labels = np.asarray(labels).reshape(2,2)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap= "YlGn",
xticklabels = classes,yticklabels = classes)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc726481450>
```



The results after fine tuning this model now has an accuracy of 93%. This seems like a rather large jump from the previous un fine tuned model and leads me to belive there may be some overfitting occuring. Fine tunning the last few layers do not tend to make this big of an increase but could be a coincidence and the model is now just predicting more accurately or more likely the model is now greatly overfitting and getting unrealistic results. However compared to the baseline model is an incrase shwoing that using a pre defined model with some fine tuning can return a more accurate result.

As well as looking at the matrix above, the model only misses a diagnosis of a paitent with Pneumonia only 2.5% of the time. This is very good as this is the worst outcome for the model but has a very low likelyhood of occuring.

InceptionResnetV2

InceptionResnetV2 is a convolutional neaural network which has been trained on millions of images to help to predict and classify images. InceptionResnetV2 is a network whoch is deeper than just inception V3 giving a higher accuracy level when modeling images. InceptionResnetV2 has been used in the classifaiction of breast cancer and has shown very promsing resluts esspecioally when some data augmentation is required to overcome some lack of data. This is why it has been used in this study to detect pneumonia in the patients.

https://link.springer.com/chapter/10.1007/978-3-319-93000-8_86

In [ ]:

```
tf.keras.backend.clear_session()

from keras.applications.inception_resnet_v2 import InceptionResNetV2

base_model = InceptionResNetV2(
  weights='imagenet',
  input_shape=(img_size, img_size, 3),
  include_top=False)

# freeze the layers
```

```
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/incept
ion_resnet_v2/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219062272/219055592 [==============================] - 2s 0us/step
219070464/219055592 [==============================] - 2s 0us/step
```

In [ ]:

```
def get_pretrained():
#Input shape = [width, height, color channels]
  inputs = layers.Input(shape=(img_size,img_size, 3))
  x = base_model(inputs)
  # Head
  x = layers.GlobalAveragePooling2D()(x)
  x = layers.Dense(128, activation='relu')(x)
  x = layers.Dropout(0.1)(x)
  #Final Layer (Output)
  output = layers.Dense(1, activation='sigmoid')(x)
  model = tf.keras.Model(inputs=[inputs], outputs=output)
  return model
```

In [ ]:

```
model_pretrained = get_pretrained()
model_pretrained.compile(loss='binary_crossentropy', optimizer = tf.keras.optimizers.Ada
m(learning_rate=0.00005), metrics='binary_accuracy')
model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 90, 90, 3)]       0

 inception_resnet_v2 (Functi  (None, 1, 1, 1536)       54336736
 onal)

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               196736

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 54,533,601
Trainable params: 196,865
Non-trainable params: 54,336,736
_________________________________________________________________
```

In [ ]:

```
history_tl = model_pretrained.fit(datagen.flow(x_train,y_train, batch_size = batch_size)
,
epochs = n_epochs , validation_data = datagen.flow(x_val,y_val) ,
callbacks = [learning_rate_reduction],
steps_per_epoch = x_train.shape[0]/batch_size,
class_weight = class_weight
)
```

```
Epoch 1/17
231/231 [==============================] - 25s 65ms/step - loss: 0.5660 - binary_accuracy
: 0.6934 - val_loss: 0.3941 - val_binary_accuracy: 0.8238 - lr: 5.0000e-05
Epoch 2/17
231/231 [==============================] - 11s 46ms/step - loss: 0.4116 - binary_accuracy
: 0.8186 - val_loss: 0.4203 - val_binary_accuracy: 0.8305 - lr: 5.0000e-05
Epoch 3/17
231/231 [==============================] - 12s 52ms/step - loss: 0.3557 - binary_accuracy
: 0.8449 - val_loss: 0.3234 - val_binary_accuracy: 0.8630 - lr: 5.0000e-05
```

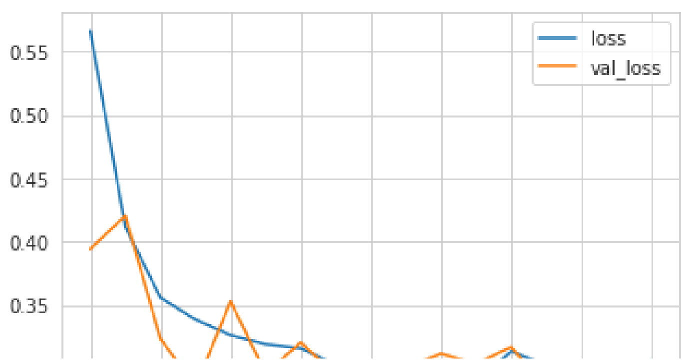
```
Epoch 4/17
231/231 [==============================] - 13s 57ms/step - loss: 0.3384 - binary_accuracy
: 0.8607 - val_loss: 0.2849 - val_binary_accuracy: 0.8927 - lr: 5.0000e-05
Epoch 5/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3262 - binary_accuracy
: 0.8648 - val_loss: 0.3529 - val_binary_accuracy: 0.8649 - lr: 5.0000e-05
Epoch 6/17
231/231 [============================>.] - ETA: 0s - loss: 0.3191 - binary_accuracy: 0.86
69
Epoch 6: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
231/231 [==============================] - 11s 46ms/step - loss: 0.3190 - binary_accuracy
: 0.8670 - val_loss: 0.2957 - val_binary_accuracy: 0.8822 - lr: 5.0000e-05
Epoch 7/17
231/231 [==============================] - 11s 46ms/step - loss: 0.3157 - binary_accuracy
: 0.8641 - val_loss: 0.3204 - val_binary_accuracy: 0.8774 - lr: 1.0000e-05
Epoch 8/17
231/231 [============================>.] - ETA: 0s - loss: 0.3028 - binary_accuracy: 0.87
48
Epoch 8: ReduceLROnPlateau reducing learning rate to 1.9999999494757505e-06.
231/231 [==============================] - 11s 46ms/step - loss: 0.3026 - binary_accuracy
: 0.8751 - val_loss: 0.2947 - val_binary_accuracy: 0.8898 - lr: 1.0000e-05
Epoch 9/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3001 - binary_accuracy
: 0.8691 - val_loss: 0.3018 - val_binary_accuracy: 0.8898 - lr: 2.0000e-06
Epoch 10/17
231/231 [============================>.] - ETA: 0s - loss: 0.2953 - binary_accuracy: 0.88
11
Epoch 10: ReduceLROnPlateau reducing learning rate to 1e-06.
231/231 [==============================] - 11s 47ms/step - loss: 0.2957 - binary_accuracy
: 0.8811 - val_loss: 0.3006 - val_binary_accuracy: 0.8841 - lr: 2.0000e-06
Epoch 11/17
231/231 [==============================] - 11s 46ms/step - loss: 0.2932 - binary_accuracy
: 0.8708 - val_loss: 0.3115 - val_binary_accuracy: 0.8860 - lr: 1.0000e-06
Epoch 12/17
231/231 [==============================] - 11s 48ms/step - loss: 0.2888 - binary_accuracy
: 0.8770 - val_loss: 0.3034 - val_binary_accuracy: 0.8726 - lr: 1.0000e-06
Epoch 13/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3133 - binary_accuracy
: 0.8698 - val_loss: 0.3168 - val_binary_accuracy: 0.8784 - lr: 1.0000e-06
Epoch 14/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3024 - binary_accuracy
: 0.8698 - val_loss: 0.2842 - val_binary_accuracy: 0.8841 - lr: 1.0000e-06
Epoch 15/17
231/231 [==============================] - 11s 46ms/step - loss: 0.3005 - binary_accuracy
: 0.8734 - val_loss: 0.2820 - val_binary_accuracy: 0.8908 - lr: 1.0000e-06
Epoch 16/17
231/231 [==============================] - 11s 46ms/step - loss: 0.2949 - binary_accuracy
: 0.8770 - val_loss: 0.2781 - val_binary_accuracy: 0.8985 - lr: 1.0000e-06
Epoch 17/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3063 - binary_accuracy
: 0.8718 - val_loss: 0.3020 - val_binary_accuracy: 0.8784 - lr: 1.0000e-06
```

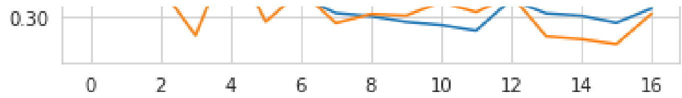
In [ ]:

```
losses = pd.DataFrame(model_pretrained.history.history)
losses[['loss','val_loss']].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc69a4bcfd0>
```



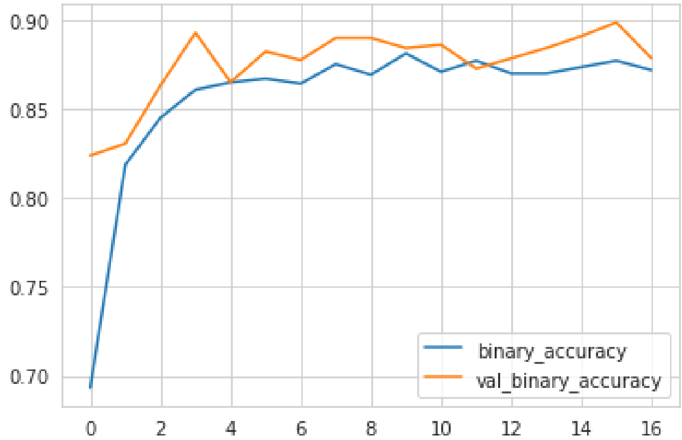


In [ ]:

```
losses[['binary_accuracy','val_binary_accuracy']].plot()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc71d949f10>
```



In [ ]:

```
print("Loss of the model is - " , model_pretrained.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model_pretrained.evaluate(x_test,y_test)[1]*100 ,
"%")
```

```
20/20 [==============================] - 4s 71ms/step - loss: 0.3463 - binary_accuracy: 0
.8686
Loss of the model is -  0.3463384509086609
20/20 [==============================] - 1s 39ms/step - loss: 0.3463 - binary_accuracy: 0
.8686
Accuracy of the model is -  86.85897588729858 %
```

InceptionResnetV2 Result:

This model has a very high accuracy initially of 87%. This shows that he transfer learning model used here has initially done very well at being able to predict wheather or not the paitent has pneumonia or not. This is a very promsining result and would be very useful in the real world with more computing power to get an even more accurate result. Increasing the epoch or image size could incrase the accuracy more.

In [ ]:

```
predictions = model_pretrained.predict(x_test)
pred_labels= np.where(predictions>0.5, 1, 0)
```

In [ ]:

```
from tensorflow.keras.models import load_model
model.save('InceptionResnetV2.h5')
#cnn_model = load_model("./CNN_PNEUMONIA.h5")

predictions = (model.predict(x_test)>0.5).astype("int32")
predictions = predictions.reshape(1,-1)[0]
predictions[:15]
```

Out[ ]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [ ]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(classification_report(y_test, predictions, target_names = ['Pneumonia(Class 0)','Normal (Class 1)']))
```

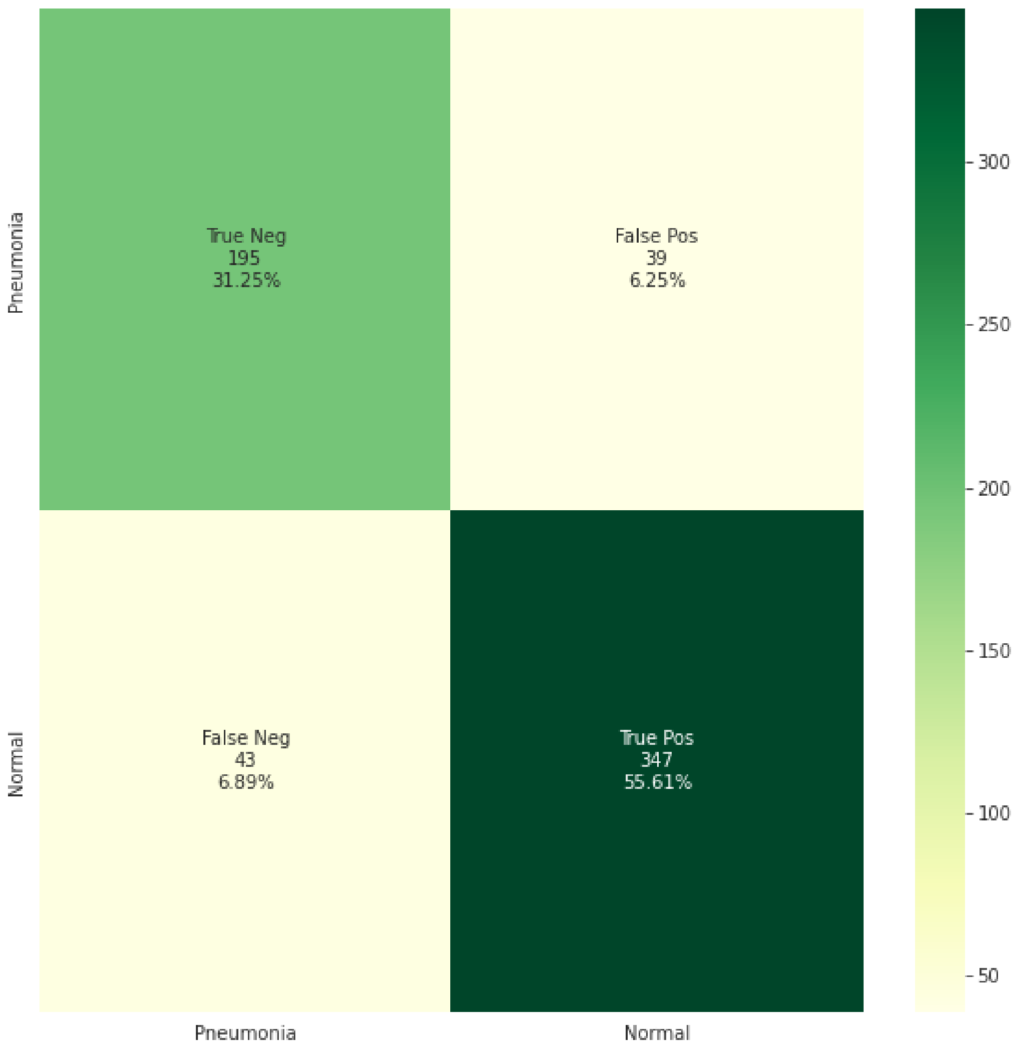
```
                    precision    recall  f1-score   support

Pneumonia(Class 0)       0.84      0.82      0.83       234
  Normal (Class 1)       0.89      0.91      0.90       390

          accuracy                           0.87       624
         macro avg       0.87      0.86      0.86       624
      weighted avg       0.87      0.87      0.87       624
```

In [ ]:

```
cf_matrix = confusion_matrix(y_test, pred_labels)
plt.figure(figsize = (10,10))
labels = ['TN','FP','FN','TP']
labels = np.asarray(labels).reshape(2,2)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap= "YlGn",
xticklabels = classes,yticklabels = classes)
```

Out[ ]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fc71d927f90>



Fine Tuning InceptionResnetV2

Below are the fine tuning done on the InceptionResnetV2 to try and incrase the accuracy of the predictions by allowing the last 10 layers of the network to become trainable and unfrozen. This will allow for a more accurate result as the model will be able to make more informed predictions.

https://medium.com/@timsennett/unfreezing-the-layers-you-want-to-fine-tune-using-transfer-learning-1bad8cb72e5d

In [ ]:

```
#Fine tunning
base_model.trainable = True
# Retrain the last 10 layers (all lower layers will be kept frozen)
for layer in base_model.layers[:-10]:
    layer.trainable = False
```

In [ ]:

```
model_pretrained.compile(loss='binary_crossentropy', optimizer = tf.keras.optimizers.Ada
m(learning_rate=0.000002), metrics='binary_accuracy')
model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 90, 90, 3)]       0

 inception_resnet_v2 (Functi  (None, 1, 1, 1536)       54336736
 onal)

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               196736

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 54,533,601
Trainable params: 4,327,649
Non-trainable params: 50,205,952
_________________________________________________________________
```

In [ ]:

```
history_ft = model_pretrained.fit(datagen.flow(x_train,y_train, batch_size = batch_size)
,
epochs = n_epochs , validation_data = datagen.flow(x_val, y_val) ,
callbacks = [learning_rate_reduction],
steps_per_epoch = x_train.shape[0]/batch_size,
class_weight = class_weight
)
```

```
Epoch 1/17
231/231 [==============================] - 23s 59ms/step - loss: 0.4378 - binary_accuracy
: 0.8550 - val_loss: 0.3232 - val_binary_accuracy: 0.8793 - lr: 2.0000e-06
Epoch 2/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3885 - binary_accuracy
: 0.8514 - val_loss: 0.3204 - val_binary_accuracy: 0.8755 - lr: 2.0000e-06
Epoch 3/17
231/231 [============================>.] - ETA: 0s - loss: 0.3740 - binary_accuracy: 0.84
91
Epoch 3: ReduceLROnPlateau reducing learning rate to 1e-06.
231/231 [==============================] - 11s 46ms/step - loss: 0.3736 - binary_accuracy
: 0.8490 - val_loss: 0.3492 - val_binary_accuracy: 0.8697 - lr: 2.0000e-06
Epoch 4/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3616 - binary_accuracy
: 0.8615 - val_loss: 0.3427 - val_binary_accuracy: 0.8707 - lr: 1.0000e-06
Epoch 5/17
```

```
231/231 [==============================] - 11s 47ms/step - loss: 0.3563 - binary_accuracy
: 0.8531 - val_loss: 0.3296 - val_binary_accuracy: 0.8764 - lr: 1.0000e-06
Epoch 6/17
231/231 [==============================] - 11s 48ms/step - loss: 0.3682 - binary_accuracy
: 0.8552 - val_loss: 0.3426 - val_binary_accuracy: 0.8726 - lr: 1.0000e-06
Epoch 7/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3464 - binary_accuracy
: 0.8543 - val_loss: 0.3107 - val_binary_accuracy: 0.8946 - lr: 1.0000e-06
Epoch 8/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3346 - binary_accuracy
: 0.8648 - val_loss: 0.3254 - val_binary_accuracy: 0.8774 - lr: 1.0000e-06
Epoch 9/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3373 - binary_accuracy
: 0.8648 - val_loss: 0.3139 - val_binary_accuracy: 0.8879 - lr: 1.0000e-06
Epoch 10/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3384 - binary_accuracy
: 0.8641 - val_loss: 0.3180 - val_binary_accuracy: 0.8774 - lr: 1.0000e-06
Epoch 11/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3375 - binary_accuracy
: 0.8658 - val_loss: 0.3360 - val_binary_accuracy: 0.8784 - lr: 1.0000e-06
Epoch 12/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3322 - binary_accuracy
: 0.8655 - val_loss: 0.3229 - val_binary_accuracy: 0.8793 - lr: 1.0000e-06
Epoch 13/17
231/231 [==============================] - 11s 46ms/step - loss: 0.3398 - binary_accuracy
: 0.8531 - val_loss: 0.3278 - val_binary_accuracy: 0.8927 - lr: 1.0000e-06
Epoch 14/17
231/231 [==============================] - 11s 46ms/step - loss: 0.3363 - binary_accuracy
: 0.8643 - val_loss: 0.3365 - val_binary_accuracy: 0.8669 - lr: 1.0000e-06
Epoch 15/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3276 - binary_accuracy
: 0.8686 - val_loss: 0.2902 - val_binary_accuracy: 0.8927 - lr: 1.0000e-06
Epoch 16/17
231/231 [==============================] - 11s 46ms/step - loss: 0.3366 - binary_accuracy
: 0.8691 - val_loss: 0.2903 - val_binary_accuracy: 0.8975 - lr: 1.0000e-06
Epoch 17/17
231/231 [==============================] - 11s 47ms/step - loss: 0.3215 - binary_accuracy
: 0.8715 - val_loss: 0.2979 - val_binary_accuracy: 0.8898 - lr: 1.0000e-06
```

In [ ]:

```
print("Loss of the model is - " , model_pretrained.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model_pretrained.evaluate(x_test,y_test)[1]*100 ,
"%")
```

```
20/20 [==============================] - 4s 44ms/step - loss: 0.3396 - binary_accuracy: 0
.8702
Loss of the model is -  0.3396260142326355
20/20 [==============================] - 1s 38ms/step - loss: 0.3396 - binary_accuracy: 0
.8702
Accuracy of the model is -  87.0192289352417 %
```

Evaluate and compare results of InceptionResnetV2 against the baseline model

After fine tunning the model it appears that the model has not managed to get more accurate as it stays the same at 87%. This could be due to the fact that the model already had a very high percentage of accuarcy and that the fine tunning did not have any effect on being able to make the model more accurate. It could also be that more/differnet fine tunning is required on this type of transfer learning to increase the accuracy of this model to try and bring the accuracy up.

In [ ]:

```
predictions = model_pretrained.predict(x_test)
pred_labels= np.where(predictions>0.5, 1, 0)
```

In [ ]:

```
from tensorflow.keras.models import load_model
```

```
model.save('InceptionResnetV2.h5')
#cnn_model = load_model("./CNN_PNEUMONIA.h5")

predictions = (model.predict(x_test)>0.5).astype("int32")
predictions = predictions.reshape(1,-1)[0]
predictions[:15]
```

Out[ ]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [ ]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test, predictions, target_names = ['Pneumonia(Class 0)','N
ormal (Class 1)']))
```

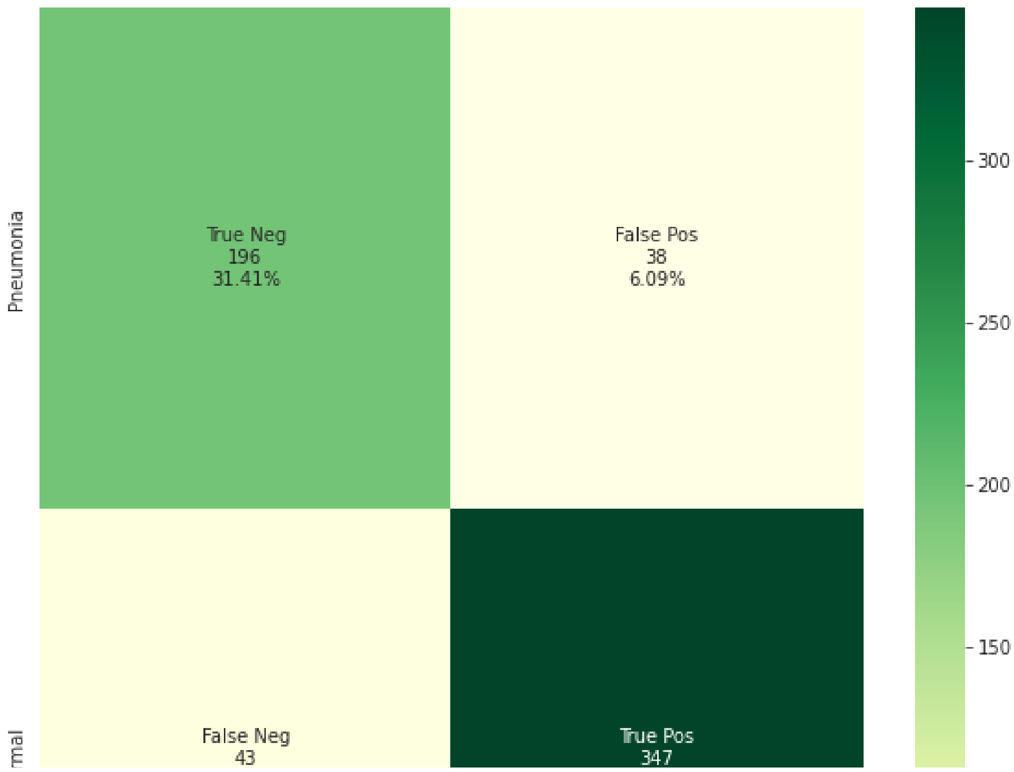
```
                    precision    recall  f1-score   support

Pneumonia(Class 0)       0.84      0.82      0.83       234
  Normal (Class 1)       0.89      0.91      0.90       390

          accuracy                           0.87       624
         macro avg       0.87      0.86      0.86       624
      weighted avg       0.87      0.87      0.87       624
```

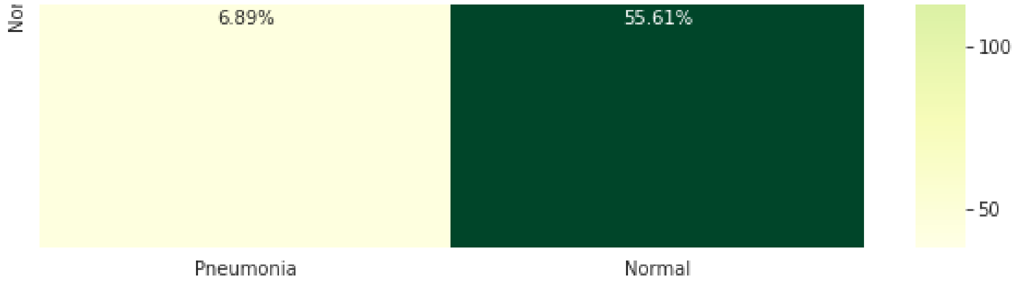
In [ ]:

```
cf_matrix = confusion_matrix(y_test, pred_labels)
plt.figure(figsize = (10,10))
labels = ['TN','FP','FN','TP']
labels = np.asarray(labels).reshape(2,2)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap= "YlGn",
xticklabels = classes,yticklabels = classes)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7bbd01b90>
```





The above matrix again shows a very good spread of results. However in this case it appears to return a flase negative 7% of the time which is higher than previous models and the baseline model slighly but is still very low. It appears all models are good at preciting if the patient has Pneumonia or not and struggles with false postives and negaitves the most.

Conclusion & Future Direction

Conclusion

The baseline model which I have implemented here has quite a high accuracy when being able to detect if pneumonia is present or not. It is extremely useful creating your own CNN model as it allows you add your own layers to the model and experiment with changing parameters. It also allows you to be able to make informed choice on what layers are required to get the best outcome. The downside of creating your own baseline model I that It is time consuming to figure out what layers are required, and which ones would benefit the model the most. It has also manged to get an accuracy of around 87% for our images which is promising.

By using a pre-defined model, it takes all of the complications of cratering your own CNN model away. It gives you a ready to use model which requires little to no changes to give you very accurate results. Along with some very minor fine tuning it can give you results of 95%+. The issue with using a pre define model is there is no customable features for the model which can mean that some are not right for the function you are trying to use them for.

Future Direction

In the future if this were to be repeated a higher image resolution would be used to help increase the accuracy of the results. This along with a higher batch size and epochs would all allow for a more accurate result. This was not possible during this study due to memory and hardware limitations but with these removed even more accurate results would be possible. More images would also be useful as having more data to test on would help to get the model to be able to identify features more easily further increasing the accuracy.