RSNA PNEUMONIA DETECTION

Team:

Aaron Pereira
Sahrash Shaqeeb
Dhone Sureendra
Vaishakh Kombilath

Project Summery

In this capstone project, the goal is to build a pneumonia detection system, to locate the position of inflammation in an image. Tissues with sparse material, such as lungs which are full of air, do not absorb the X-rays and appear black in the image. Dense tissues such as bones absorb X-rays and appear white in the image. While we are theoretically detecting “lung capacities”, there are lung capacities that are not pneumonia related. In the data, some of these are labeled “Not Normal No Lung Opacity”. This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia. Dicom original images: - Medical images are stored in a special format called DICOM files (*.dcm). They contain a combination of header meta data as well as underlying raw image arrays for pixel data.

Problem Statement

In this capstone project, the goal is to build a pneumonia detection system, to locate the position of inflammation in an image. Tissues with sparse material, such as lungs which are full of air, do not absorb the X-rays and appear black in the image. Dense tissues such as bones absorb X-rays and appear white in the image. While we are theoretically detecting “lung capacities”, there are lung capacities that are not pneumonia related. In the data, some of these are labeled “**Not Normal No Lung Opacity**”. This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia.

Objectives

- ✔ Learn to how to do build an Object Detection Model
- ✔ Use transfer learning to fine-tune a model.
- ✔ Learn to set the optimizer, loss functions, epochs, learning rate, batch size, check pointing, early stopping etc
- ✔ Read different research papers of given domain to obtain the knowledge of advanced models for the given problem.

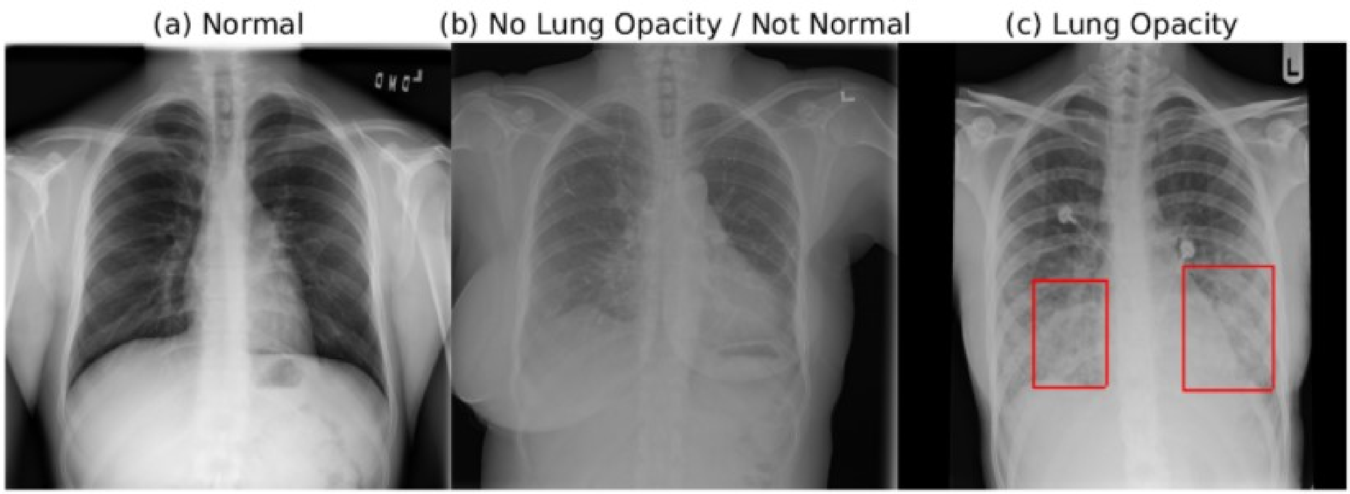
Methodologies

1. Data set preparation
   - Cleaning
   - Visualization
   - Statistical analysis
2. Image Augmentation
   - Image conversion .dcm to .png
   - Re sizing as per the model preprocessing requirement
3. Model Building
   - Model initialization
   - Compiling/Fit the data
4. Testing accuracy
   - Classification report
   - Model performance graph

Dataset/ EDA

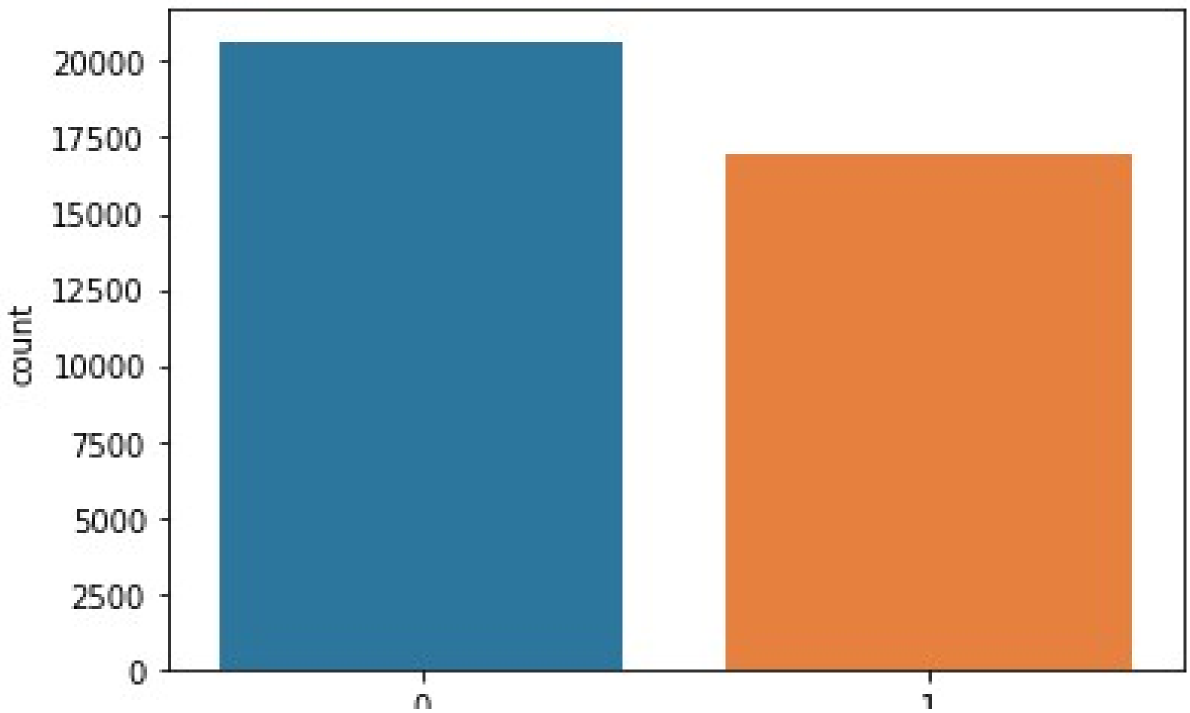
The labeled data-set of the chest X-Ray (CXR) images and patients meta data was publicly provided for the challenge by the US National Institutes of Health Clinical Center. The data-set is available on kaggle platform.

The database comprises frontal-view X-ray images from 26684 unique patients. Each image is labeled with one of three different classes from the associated radiological reports: ”Normal”, ”No Lung Opacity / Not Normal”, ”Lung Opacity”. Fig. 1 shows examples of all three classes CXRs labeled with bounding boxes for unhealthy patients.

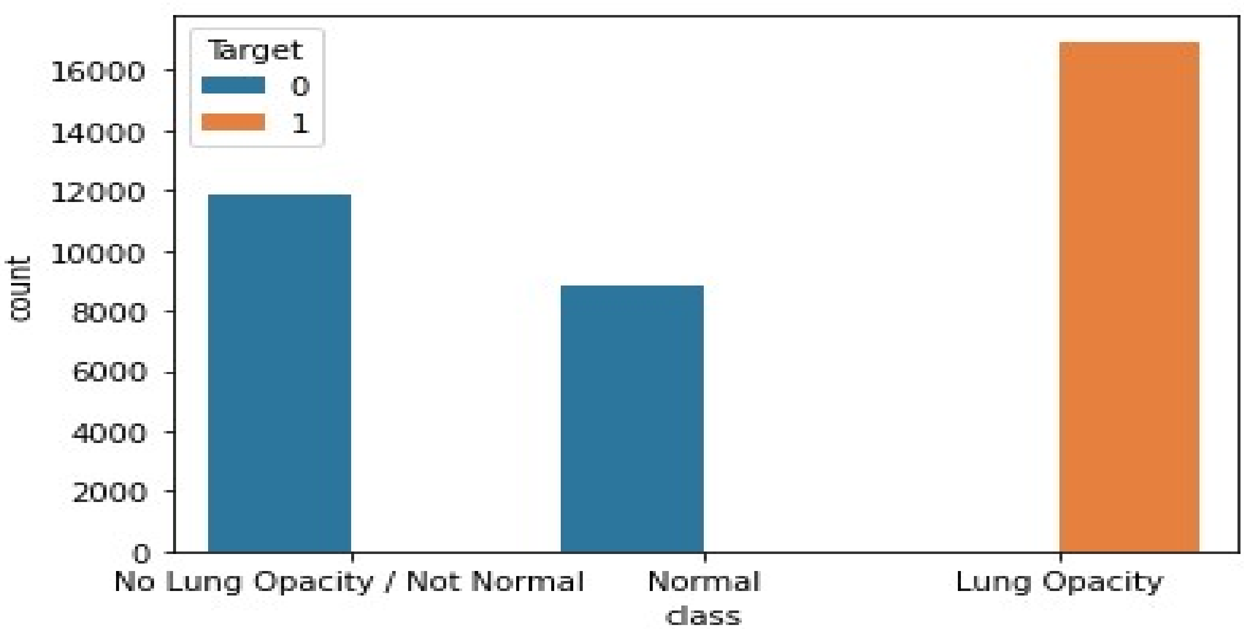


Since we have 3 classes, the distribution among total data is as shown below.

<matplotlib.axes._subplots.AxesSubplot at 0x7fcbbdc4c400>



<matplotlib.axes._subplots.AxesSubplot at 0x7fcbbdbae550>



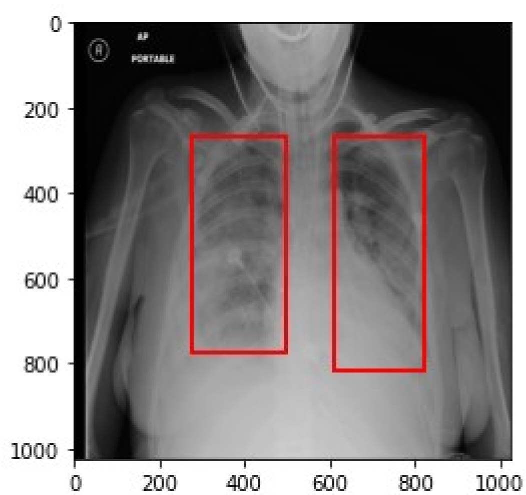
Checking Distribution of Target

From the CSV data, the target column gives the information of the whether the patient is having pneumonia positive or not by **1&0.** The distribution of the target is as shown below. Total positive cases are 32% and Negative cases are 68%.

Initial Observations

- Null values are present only with bounding box data I the given CSV.
- All the bounding box null values are associated with target **0**
- Each patient ID is associated with a single class or target
- Many of the patient id's are associated with more than one bounding boxes

number_of_boxes	number_of_patientIDs_per_boxes
1	23286
2	3266
3	119
4	13



- All positive cases are associated with target **1** only.
- The X-ray images are in .dcm format with a resolution of 1024.
- We have total 30227, in that missing value samples 20672 and 9555.
- We have unique patient id's 26684 and we have duplicate patient id's of 3543.

Sample patient information from the .dcm image is as shown below

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 202
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: "1"
(0020, 0013) Instance Number                     IS: "1"
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.14300000000000002, 0.14300000000000002]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 142006 elements
```

Steps Followed in Data Preparation

- ✔ Understanding the data with a brief on train/test labels and respective class info
- ✔ Look at the first five rows of both the .csv files(train and test).
- ✔ Identify how are classes and target distributed
- ✔ Check the number of patients with 1, 2, ... bounding boxes
- ✔ Read and extract meta data from dicom files
- ✔ Perform analysis on some of the features from dicom files
- ✔ Check some random images from the training dataset
- ✔ Draw insights from the data at various stages of EDA
- ✔ Visualize some random masks generated

From the dataset, we have classification and regression statement. Whereas the classification part comes with predicting pneumonia positive or negative and the regression part has to predict the area which opacity has found and draw the bounding box.

Image Extraction

Image extraction involves saving image path in input training variable along with making bounding dependency variable & target variable. We handled Null/Nan bounding box variable as zero.

```
def extractImages(foldername,data):
    X_image_train = []
    y_image_train = np.zeros((len(data),4))
    y_train_Target = np.zeros((len(data),1))
    for index,row in data.iterrows():
        name = row[0]
        x1 = int(row[1])
        y1 = int(row[2])
        path = os.path.join(foldername,name)
        path = path+'.png'
        img = cv2.imread(path)
        image_width = img.shape[1]
        image_height = img.shape[0]
        width = int(row[3])
        height = int(row[4])
        target = int(row[5])
        if width != 0 :
            y_image_train[index,0] = x1* image_size/image_width
            y_image_train[index,1] = y1* image_size/image_height
            y_image_train[index,2] = ((width+x1)-x1)* image_size/image_width
            y_image_train[index,3] = ((height+y1)-y1)* image_size/image_width
        else:
            y_image_train[index,0] = 0
            y_image_train[index,1] = 0
            y_image_train[index,2] = 0
            y_image_train[index,3] = 0
        y_train_Target[index] = target
        X_image_train.append(path)

    return (X_image_train,y_image_train,y_train_Target)
```

Image Preprocessing

Pre-processing involves scaling the image to desired size for the selected model. We used PIL package to read, re-size and converting image to RGB(to make 3 channel).

```
def preprocessImage(data):
    processed_data = []
    for i,f in enumerate(data):
        img = Image.open(f)
        img = img.resize((image_size, image_size)) # Resize image
        img = img.convert('RGB')
        processed_data.append(preprocess_input(np.array(img, dtype=np.float32)))
        img.close()
    return processed_data
```

Model Selection

We have prepared two basic models using Resnet50 and VGG16 architecture with pre-trained “imagenet” weights. Here we are only training top layers which is defined by our self. The model has an input shape of images width, height and channels 224, 224, 3 respectively.

The model is fit on 10000 input samples & validation done on 2000 samples.

VGG16 - CNN Model

Creating Model

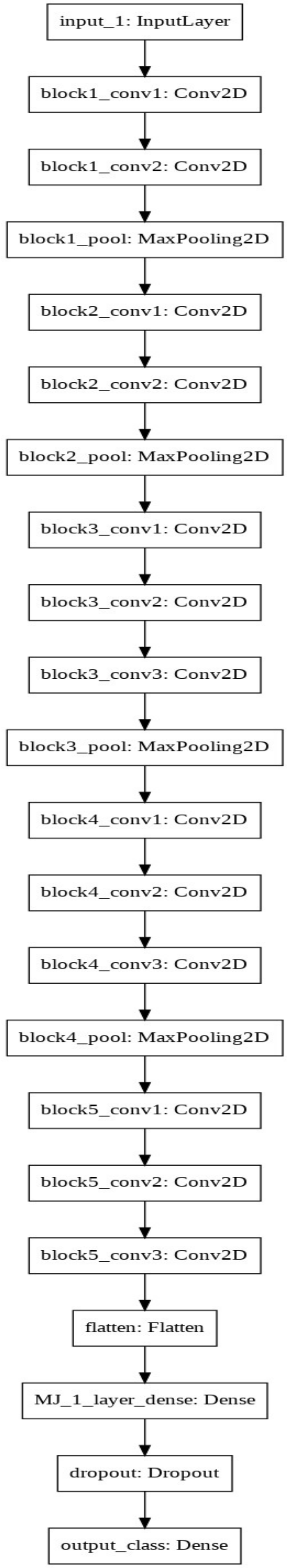
```python
# Here, we are using VGG16 model
# And using input shape (224,224,3) 3 channels.
# And using 'imagenet weights'.
# This is the basic classification model.
def createModel(trainBaseModel=True):
    inputShape = (image_size,image_size,3)

    basemodel = VGG16(include_top=False,input_shape=inputShape,weights='imagenet')

    for layer in basemodel.layers:
        layer.trainable = trainBaseModel

    basemodel_output =  basemodel.get_layer('block5_conv3').output

    flat_class = Flatten()(basemodel_output)
    dense = Dense(512,activation='relu',name='MJ_1_layer_dense')(flat_class)
    drop = Dropout(0.2)(dense)
    output_class = Dense(2,activation='softmax',name='output_class')(drop)

    return Model(inputs=basemodel.input, outputs=[output_class])
```

Model Summery

input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
flatten (Flatten)	(None, 100352)	0
MJ_1_layer_dense (Dense)	(None, 512)	51380736
dropout (Dropout)	(None, 512)	0
output_class (Dense)	(None, 2)	1026

```
Total params: 66,096,450
Trainable params: 51,381,762
Non-trainable params: 14,714,688
```

VGG16 Flowchart



Model Performance

The model is fit on 10000 input samples & validation done on 2000 samples.

- Defining loss function & fitting the model

```
[ ] # Use earlystcheckpoint
    checkpoint = ModelCheckpoint("model-{loss:.2f}.h5", monitor="loss", verbose=1, save_best_only=True,
                                 save_weights_only=True, mode="min", save_freq='epoch')

    # Use earlystopping
    stop = EarlyStopping(patience=2, mode="min",min_delta=0.01,monitor='output_class_loss' )
```

```
[ ] model.compile(loss="categorical_crossentropy", optimizer="adam", metrics="accuracy")

    # Fit the model
    history = model.fit(X_image_train_temp,y_target_train, epochs=20, batch_size=128,callbacks=[stop])
```

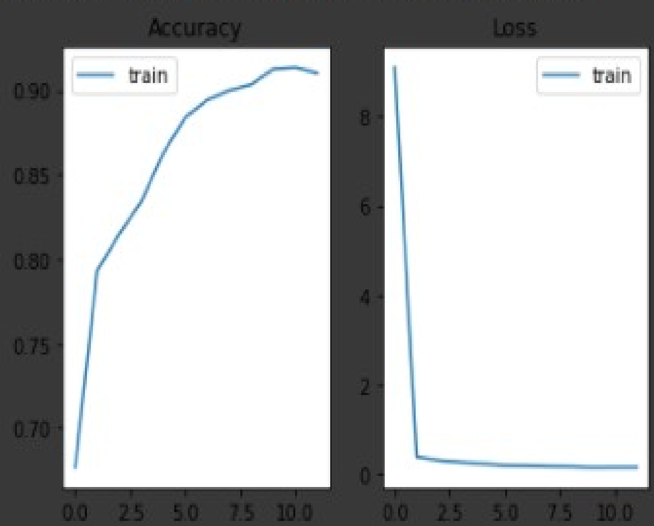
Model Performance Visualization

```
Epoch 10/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0787 - accuracy: 0.9727WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0787 - accuracy: 0.9726
Epoch 11/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0356 - accuracy: 0.9872WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0356 - accuracy: 0.9872
Epoch 12/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0220 - accuracy: 0.9909WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0220 - accuracy: 0.9909
Epoch 13/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0256 - accuracy: 0.9921WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 242ms/step - loss: 0.0256 - accuracy: 0.9921
Epoch 14/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0234 - accuracy: 0.9921WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0239 - accuracy: 0.9920
Epoch 15/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0253 - accuracy: 0.9919WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 242ms/step - loss: 0.0253 - accuracy: 0.9919
Epoch 16/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0266 - accuracy: 0.9896WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0265 - accuracy: 0.9896
Epoch 17/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0230 - accuracy: 0.9912WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0229 - accuracy: 0.9912
Epoch 18/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0190 - accuracy: 0.9923WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0190 - accuracy: 0.9923
Epoch 19/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0198 - accuracy: 0.9922WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 241ms/step - loss: 0.0199 - accuracy: 0.9922
Epoch 20/20
78/79 [==========================>.] - ETA: 0s - loss: 0.0241 - accuracy: 0.9908WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [============================] - 19s 242ms/step - loss: 0.0240 - accuracy: 0.9908
```

Model Performance Visualization

```python
fig,ax = plt.subplots(1,2)
ax[0].set_title('Accuracy')
ax[0].plot(history.history['output_class_accuracy'],label='train')
# ax[0].plot(history.history['val_accuracy'],label='test')
ax[0].legend()

ax[1].set_title('Loss')
ax[1].plot(history.history['output_class_loss'],label='train')
# ax[1].plot(history.history['val_loss'],label='test')
ax[1].legend()
# By checking the accuracy and loss plots, suggest that the model has good fit on the problem.
```

```
<matplotlib.legend.Legend at 0x7fdfdd673978>
```



RESNET 50 - CNN Model

Creating Model

```python
# Here, we are using resnet model
# And using input shape (224,224,3) 3 channels.
# And using 'imagenet weights'.
# This is the basic classification model.
def createModelResnet(trainBaseModel=True):
    inputShape = (image_size,image_size,3)

    basemodel = ResNet50(include_top=False,input_shape=inputShape)

    for layer in basemodel.layers:
        layer.trainable = trainBaseModel

    basemodel_output =  basemodel.get_layer('conv5_block3_3_conv').output

    flat_class = Flatten()(basemodel_output)
    dense = Dense(512,activation='relu',name='MJ_1_layer_dense')(flat_class)
    drop = Dropout(0.2)(dense)
    output_class = Dense(2,activation='softmax',name='output_class')(drop)

    return Model(inputs=basemodel.input, outputs=[output_class])
```

Model Summery

- Final blocks shown

```
conv5_block1_out (Activation)   (None, 7, 7, 2048)   0          conv5_block1_add[0][0]

conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)    1049088    conv5_block1_out[0][0]

conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)    2048       conv5_block2_1_conv[0][0]

conv5_block2_1_relu (Activation (None, 7, 7, 512)    0          conv5_block2_1_bn[0][0]

conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)    2359808    conv5_block2_1_relu[0][0]

conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)    2048       conv5_block2_2_conv[0][0]

conv5_block2_2_relu (Activation (None, 7, 7, 512)    0          conv5_block2_2_bn[0][0]

conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624    conv5_block2_2_relu[0][0]

conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048)   8192       conv5_block2_3_conv[0][0]

conv5_block2_add (Add)          (None, 7, 7, 2048)   0          conv5_block1_out[0][0]
                                                                conv5_block2_3_bn[0][0]

conv5_block2_out (Activation)   (None, 7, 7, 2048)   0          conv5_block2_add[0][0]

conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088    conv5_block2_out[0][0]

conv5_block3_1_bn (BatchNormali (None, 7, 7, 512)    2048       conv5_block3_1_conv[0][0]

conv5_block3_1_relu (Activation (None, 7, 7, 512)    0          conv5_block3_1_bn[0][0]

conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808    conv5_block3_1_relu[0][0]

conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048       conv5_block3_2_conv[0][0]

conv5_block3_2_relu (Activation (None, 7, 7, 512)    0          conv5_block3_2_bn[0][0]

conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624    conv5_block3_2_relu[0][0]

flatten (Flatten)               (None, 100352)       0          conv5_block3_3_conv[0][0]

MJ_1_layer_dense (Dense)        (None, 512)          51380736   flatten[0][0]

dropout (Dropout)               (None, 512)          0          MJ_1_layer_dense[0][0]

output_class (Dense)            (None, 2)            1026       dropout[0][0]
==================================================================================================
Total params: 74,961,282
Trainable params: 51,381,762
Non-trainable params: 23,579,520
```

Model Performance

The model is fit on 10000 input samples & validation done on 2000 samples.

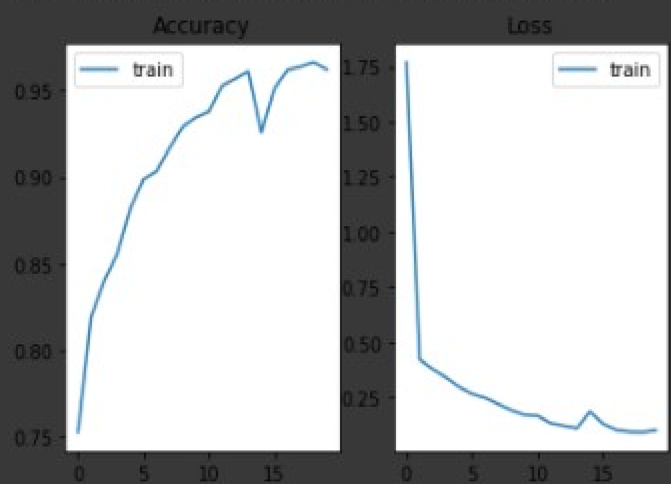
```python
# Use earlystcheckpoint
checkpoint = ModelCheckpoint("model-{loss:.2f}.h5", monitor="loss", verbose=1, save_best_only=True,
                             save_weights_only=True, mode="min", save_freq='epoch')

# Use earlystopping
stop = EarlyStopping(patience=2, mode="min",min_delta=0.01,monitor='output_class_loss' )
```

```
Epoch 11/20
79/79 [==============================] - ETA: 0s - loss: 0.1661 - accuracy: 0.9378WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 397ms/step - loss: 0.1661 - accuracy: 0.9378
Epoch 12/20
79/79 [==============================] - ETA: 0s - loss: 0.1322 - accuracy: 0.9527WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 396ms/step - loss: 0.1322 - accuracy: 0.9527
Epoch 13/20
79/79 [==============================] - ETA: 0s - loss: 0.1208 - accuracy: 0.9567WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 397ms/step - loss: 0.1208 - accuracy: 0.9567
Epoch 14/20
79/79 [==============================] - ETA: 0s - loss: 0.1085 - accuracy: 0.9610WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 398ms/step - loss: 0.1085 - accuracy: 0.9610
Epoch 15/20
79/79 [==============================] - ETA: 0s - loss: 0.1843 - accuracy: 0.9256WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 398ms/step - loss: 0.1843 - accuracy: 0.9256
Epoch 16/20
79/79 [==============================] - ETA: 0s - loss: 0.1299 - accuracy: 0.9511WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 398ms/step - loss: 0.1299 - accuracy: 0.9511
Epoch 17/20
79/79 [==============================] - ETA: 0s - loss: 0.1006 - accuracy: 0.9618WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 32s 399ms/step - loss: 0.1006 - accuracy: 0.9618
Epoch 18/20
79/79 [==============================] - ETA: 0s - loss: 0.0932 - accuracy: 0.9638WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 398ms/step - loss: 0.0932 - accuracy: 0.9638
Epoch 19/20
79/79 [==============================] - ETA: 0s - loss: 0.0912 - accuracy: 0.9663WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 398ms/step - loss: 0.0912 - accuracy: 0.9663
Epoch 20/20
79/79 [==============================] - ETA: 0s - loss: 0.0999 - accuracy: 0.9621WARNING:tensorflow:Early stopping conditioned on metric `output_class_loss` which is not available. Available metrics are: loss,accuracy
79/79 [==============================] - 31s 398ms/step - loss: 0.0999 - accuracy: 0.9621
```

Model Performance Visualization

```
fig,ax = plt.subplots(1,2)
ax[0].set_title('Accuracy')
ax[0].plot(history_resnet.history['accuracy'],label='train')
ax[0].legend()

ax[1].set_title('Loss')
ax[1].plot(history_resnet.history['loss'],label='train')
ax[1].legend()
# By checking the accuracy and loss plots, suggest that the model has good fit on the problem.
```

```
<matplotlib.legend.Legend at 0x7f82660607b8>
```



VGG16 vs. ResNet50(Accuracy Comparison)

VGG16 Model Evaluation

```
# Evaluate model
model.evaluate(X_image_test_temp,y_target_test)
```

```
63/63 [==============================] - 4s 63ms/step - loss: 1.4054 - accuracy: 0.7995
[1.4053908586502075, 0.7994999885559082]
```

Output classification accuracy: 80%

RESNET50 Model Evaluation

```
# Evaluate model
resnet_model.evaluate(X_image_test_resnet,y_target_test_resnet)
```

```
63/63 [==============================] - 6s 100ms/step - loss: 0.7435 - accuracy: 0.7770
[0.7435166239738464, 0.7770000100135803]
```

Output classification accuracy: 77%

Selecting model for final prediction

From the above comparison it is clear that VGG16 gives better accuracy in classification compared to RESNET50. So we are selecting VGG16 for the final prediction.

VGG-16 Prediction

```
def preditction(imagename):
    imageFolderPath = "/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng"

    filepath = os.path.join(imageFolderPath,imagename)
    unscaled = cv2.imread(filepath)
    image_height, image_width, _ = unscaled.shape
    image = cv2.resize(unscaled, (image_size, image_size)) # Rescaled image to run the network
    feat_scaled = preprocess_input(np.array(image, dtype=np.float32))

    result = model.predict(x=np.array([feat_scaled]))
    print(f'Class is : {result}')
```

```
# VGG16 testing
source_folder_path = "/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng"
files = os.listdir(source_folder_path)
for index in range(20,30):
  file = files[index]
  imagPath = os.path.join(source_folder_path,file)
  print(imagPath)
  preditction(imagPath)
```

```
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/265dd221-9049-4bca-b5c0-4118dafa55c5.png
Class is : [[0.99733835 0.00266162]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/265ef9f1-3a21-4c9e-a8fe-740d8fae99f5.png
Class is : [[9.9999988e-01 1.5446045e-07]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/265f1a4a-fee4-447e-9709-0fff15f2255b.png
Class is : [[9.999684e-01 3.156404e-05]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/265fc72c-f5f5-41bb-ac15-7ced161736df.png
Class is : [[9.9992275e-01 7.7258235e-05]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/26636455-c98d-49a1-8b48-7025f535f982.png
Class is : [[9.9999750e-01 2.4779479e-06]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/2664366f-4f04-49e1-ab20-19b9173f23bc.png
Class is : [[0.32920593 0.670794  ]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/266490ca-ce52-4f5b-92ff-082dae7967c0.png
Class is : [[0.4772196 0.5227804]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/267147a0-2320-4a54-85e8-a3950f0672b8.png
Class is : [[9.9999905e-01 9.4643684e-07]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/26729371-ee9c-403b-a92a-23ee1bb0bb9b.png
Class is : [[9.999999e-01 7.642126e-08]]
/content/gdrive/MyDrive/AIML/Capstone/Pneumonia_Detection/liveTestPng/2676fc9d-7ace-4896-b698-17fc68131851.png
Class is : [[0.01775745 0.9822426 ]]
```

Conclusion

We have analyzed all 30k images and we got to know that duplicated patient ids and we have considered this as different samples for the model input. In model building process we choose to go for transfer learning. So in that we choose two models VGG16 and ResNet50 based on the performance compared we have found VGG16 is better than that ResNet50.