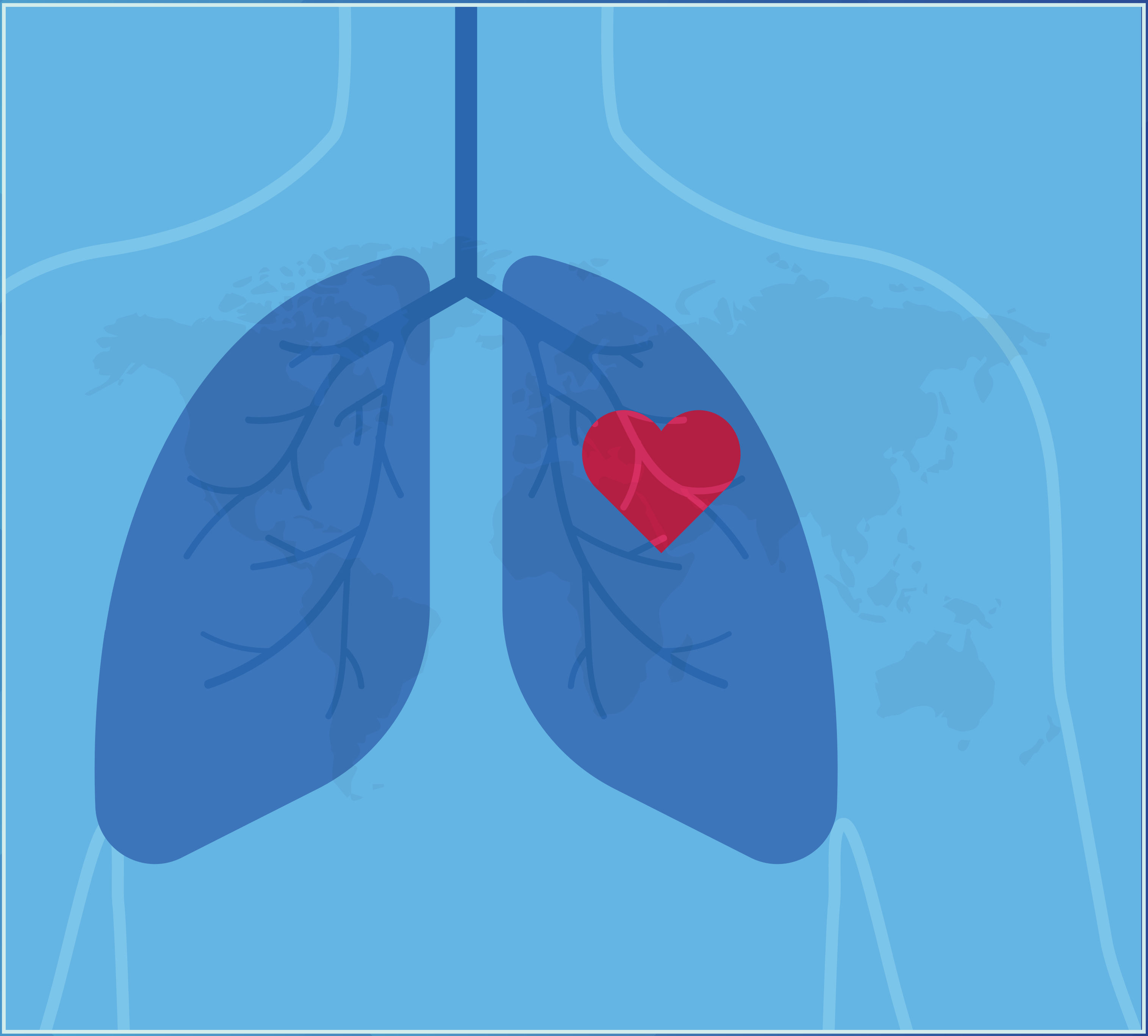
Pneumonia Detection Challenge
Computer Vision – Oct 2023



A Capstone Project

ComputerVision

Pneumonia Detection Challenge

A Capstone Project

Submitted by:

ABHAY KUMAR SINGH

RAGHAV RAMESH

ISHITA KONAR

VIJAY R

In fulfillment of the requirements for the award of

Post Graduate Program in

Artificial Intelligence & Machine Learning

from:

Great Learning

&

TEXAS McCombs

The University of Texas at Austin

Mentor: Neha Baranwal

21st October 2023

CONTENTS

Milestone 1:

1. Final Report Expectations

1.1 Expectations For Final Report submission

1. Summary of problem statement, data and findings Every good abstract describes briefly what was intended at the outset, and summarizes findings and implications.

2. Overview of the final process Briefly describe your problem methodology. Include information about the salient features of your data, data pre-processing steps, the algorithms you used and how you combined techniques.

3. Step-by-step walk through the solution Describe the steps you took to solve the problem. What did you find at each stage, and how did it inform the next steps? Build up to the final solution.

4. Model evaluation Describe the final model in detail. What was the objective, what parameters were prominent, and how did you evaluate the success of your models?

5. Comparison to benchmark How does your final solution compare to the benchmark you laid out at the outset? Did you improve on the benchmark? Why or why not?

6. Visualizations In addition to quantifying your model and the solution, please include all relevant visualizations that support the ideas/insights that you gleaned from the data.

7. Implications How does your solution affect the problem in the domain or business? What recommendations would you make, and with what level of confidence?

8. Limitations What are the limitations of your solution? Where does your model fall short in the real world? What can you do to enhance the solution?

9. Closing Reflections What have you learned from the process? What you do differently next time?

2. Methodology

We initiated our project with a binary classification approach. For the interim submission, we explored three distinct models:

Base CNN Model
Modified Version of our Base CNN
Model Based on the MobileNet Architecture

As we progressed towards the final submission, we focused on refining our base models that were initially developed in Milestone 1. This refinement involved the application of a range of advanced techniques aimed at enhancing model performance. These techniques encompassed:

Image Augmentation: We harnessed image augmentation methodologies to diversify our training dataset. This approach not only bolstered model generalization but also mitigated overfitting tendencies, resulting in more robust models.
Under sampling: To effectively address class imbalance, we employed under sampling techniques. By ensuring that our models learned from all classes, we were able to improve overall classification performance.
Hyperparameter Tuning: Hyperparameter tuning was carried out, where we systematically optimized model parameters. This enabled us to identify the ideal combination of hyperparameters that maximized model performance.
Transfer Learning: Leveraging the power of transfer learning, we applied pretrained models to our task. This strategy allowed us to benefit from feature extraction capabilities and expedited model convergence.

On the object detection front, we implemented the following models to tackle object detection challenges:

R-CNN (Region-Based Convolutional Neural Network)

Faster R-CNN: A more optimized version of the R-CNN, known for its efficiency in object detection. Segmentation: A specialized model for image segmentation tasks.

These diverse approaches were employed to ensure that our object detection pipeline was capable of accurately identifying and localizing objects of interest within images.

To avoid constant crashes on Google Colab during model code execution, the team opted to run our entire notebook on local laptop having higher memory capacity. As a result, all necessary data are loaded from local directories from laptop for a smoother and uninterrupted execution. This decision allowed us to work efficiently without interruptions caused by Colab crashes. The same has been updated in Google Colab notebook as well.

Abstract

In the Health Care dataset, some of the features 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 metadata as well as underlying raw image arrays for pixel data.

Pneumonia remains a global health concern, with timely and accurate diagnosis being crucial for effective patient management. This abstract introduces a state-of-the-art approach for pneumonia detection utilizing deep learning techniques in conjunction with medical imaging. The proposed system leverages convolutional neural networks (CNNs) to analyze chest radiographs, aiming to improve the accuracy and efficiency of pneumonia diagnosis.

The process begins with the collection of a diverse dataset comprising chest X-rays from both pneumonia-positive and pneumonia-negative cases. The dataset is carefully curated and annotated to ensure high-quality training and validation data. A pre-processing step involves image enhancement and normalization to enhance the network's ability to extract meaningful features.

Our deep learning model is designed to automatically extract relevant patterns and features from the chest radiographs. Leveraging transfer learning from pre-trained networks, such as ResNet or Inception, the model fine-tunes its parameters on the pneumonia dataset.

We were able to use some of the latest libraries to help fasttrack the training and development of models like Dense, Flatten, this may allow us to classify images based on output from convolutional layers.

The objective of this project is to help users to investigate the diagnostic accuracy of a deep learning model for the detection of pneumonia from chest X-ray images, aiming to improve early disease identification and patient care.

Problem Statement

Computer vision can be used in health care for identifying diseases. In Pneumonia detection we need to detect Inflammation of the lungs. In this project, you're required to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, your algorithm needs to automatically locate lung opacities on chest radiographs

In the dataset, some of the features 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 metadata as well as underlying raw image arrays for pixel data.

DOMAIN

Health Care

CONTEXT

Computer vision can be used in health care for identifying diseases. In Pneumonia detection we need to detect Inflammation of the lungs. In this challenge, you're required to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, your algorithm needs to automatically locate lung opacities on chest radiographs.

DATA DESCRIPTION

In the dataset, some of the features 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 metadata as well as underlying raw image arrays for pixel data.

Dataset has been attached along with this project. Please use the same for this capstone project.

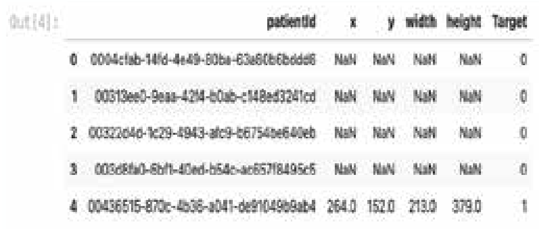
PROJECT OBJECTIVE

Design a DL based algorithm for detecting pneumonia.

Data Analysis, EDA & Preprocessing

Step 1. Import the data.

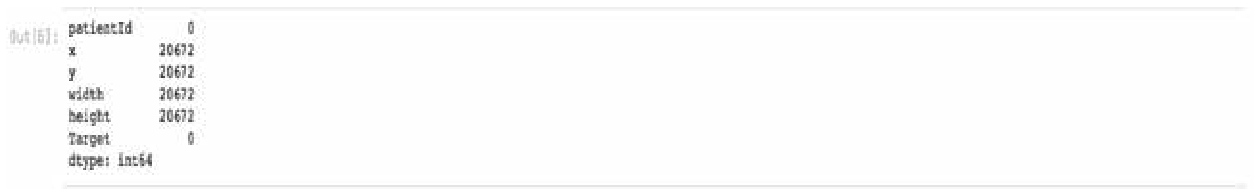
# importing labels info


Out[4]:

	patientId	x	y	width	height	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1

1) Dataset has 30227 rows & 6 Columns with no missing values.

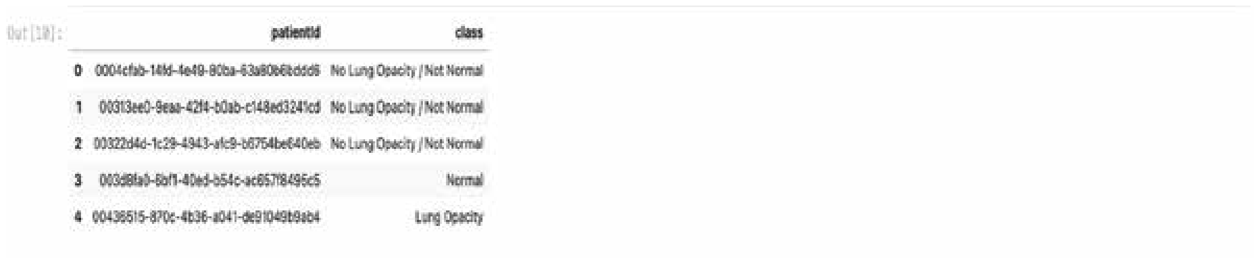
Checking for the Null-Values



```
Out[6]: patientId        0
        x            20672
        y            20672
        width        20672
        height       20672
        Target           0
        dtype: int64
```

1) There was some null value which will be dropped from the dataset in future process.

Importing detailed class info



Out[10]:

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity

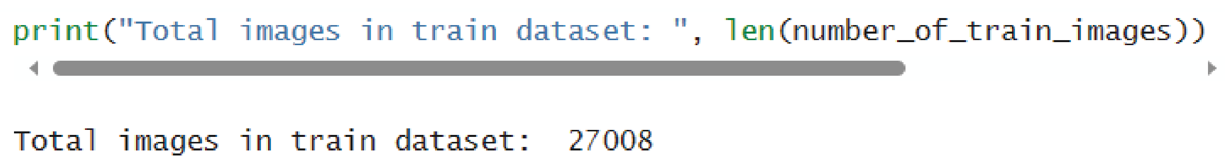
1) Dataset has 30227 rows & 2 Columns with no missing values.

Checking for the Null-Values

```
patientId    0
class        0
dtype: int64
```

1) We can’t find any null-values, so we can process future steps.

# defining image path and counting the number of images we have in train image set

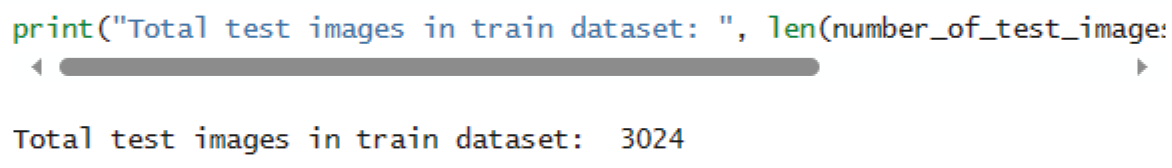


```
print("Total images in train dataset: ", len(number_of_train_images))
```

```
Total images in train dataset:  27008
```

1) Total number of images we have in train image set are 27008.

# defining test image path and counting the number of images we have in test image set

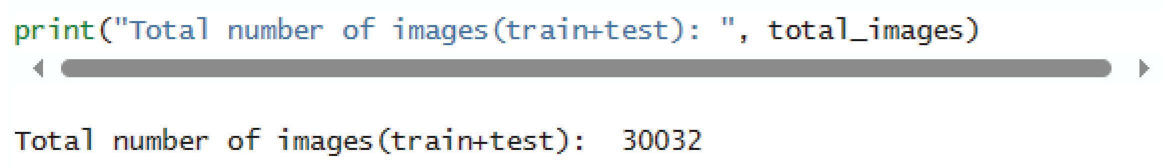


```
print("Total test images in train dataset: ", len(number_of_test_image
```

```
Total test images in train dataset:  3024
```

1) Total number of images we have in test image set are 3024.

# sum up the train and test images

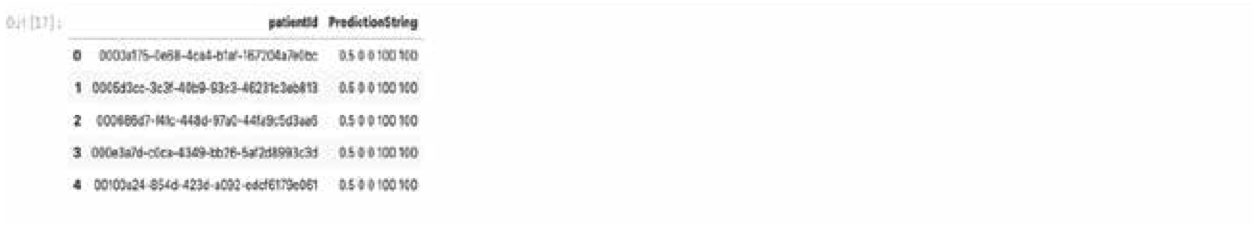


```
print("Total number of images(train+test): ", total_images)
```

```
Total number of images(train+test):  30032
```

1) Assuming all the labels and annotations are in the same CSV files for train set and test set. Because the sum of total images from train and test are very close to the total of detailed class info and labels CSV.

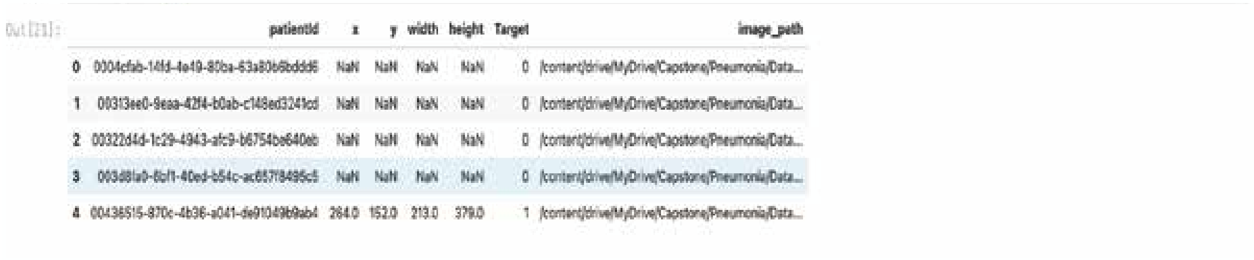
# importing submission sample



```
Out [17]:
                                 patientId  PredictionString
0  0000a175-0e68-4ca4-b1af-167204a7e0bc  0.5 0 0 100 100
1  0005d3cc-3c3f-40b9-93c3-46231c3eb813  0.5 0 0 100 100
2  000686d7-f4fc-448d-97a0-44fa9c5d3aa6  0.5 0 0 100 100
3  000e3a7d-c0ca-4349-bb26-5af2d8993c3d  0.5 0 0 100 100
4  00100a24-854d-423d-a092-edcf6179e061  0.5 0 0 100 100
```

Step 2. Map training and testing images to its classes.

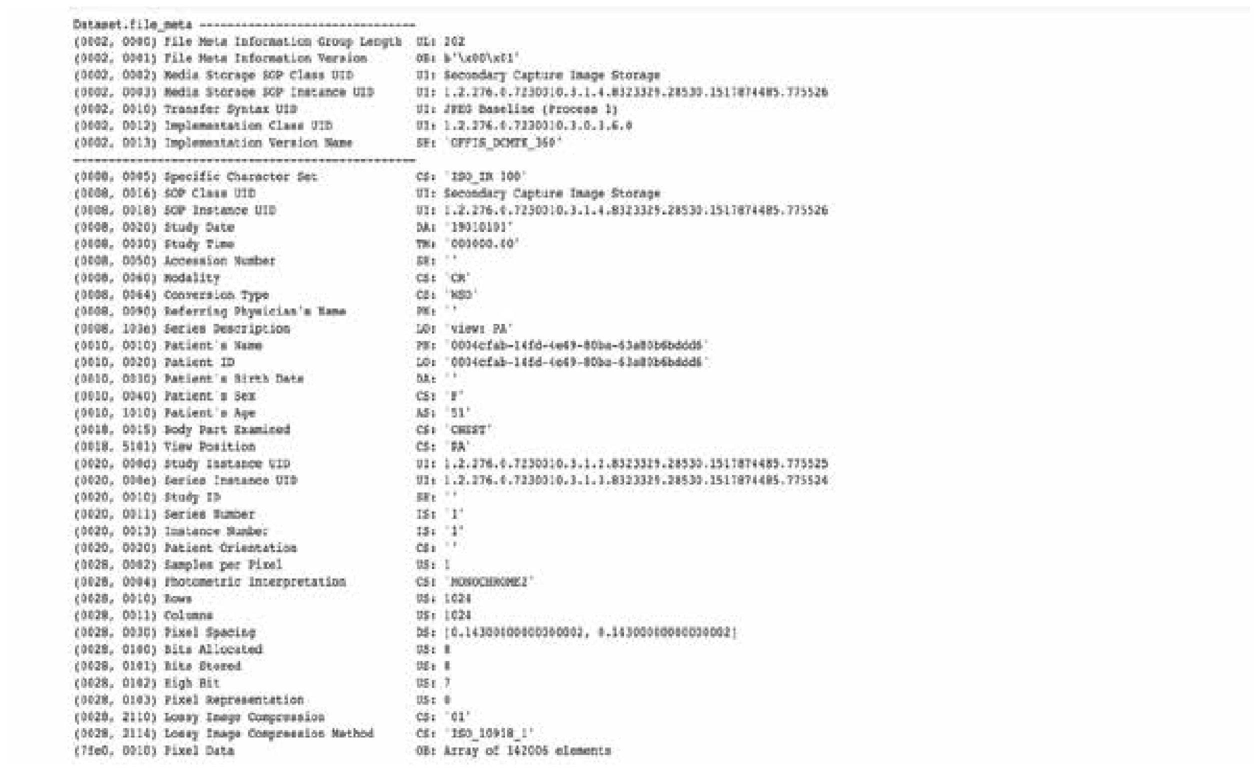
# add another column image path to pneumonia dataframe to combine image paths with the classes



```
Out [21]:
                                 patientId      x      y  width  height  Target                                 image_path
0  0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN       0  /content/drive/MyDrive/Capstone/Pneumonia/Data...
1  00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN       0  /content/drive/MyDrive/Capstone/Pneumonia/Data...
2  00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN       0  /content/drive/MyDrive/Capstone/Pneumonia/Data...
3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0  /content/drive/MyDrive/Capstone/Pneumonia/Data...
4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1  /content/drive/MyDrive/Capstone/Pneumonia/Data...
```

1) Columns added successfully are shown above.

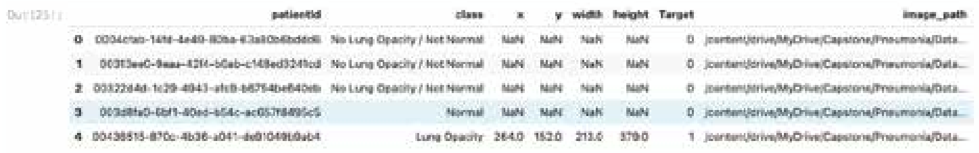
# reading Dicom data from the pneumonia dataframe



```
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
```

Step 3. Map training and testing images to its annotations.

# mapping training and testing images with the annotations by combining labels_df and detailed_info_df



Out[25]:

	patientId	class	x	y	width	height	Target	image_path
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal	NaN	NaN	NaN	NaN	0	/content/drive/MyDrive/Capstone/Pneumonia/Data...
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal	NaN	NaN	NaN	NaN	0	/content/drive/MyDrive/Capstone/Pneumonia/Data...
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal	NaN	NaN	NaN	NaN	0	/content/drive/MyDrive/Capstone/Pneumonia/Data...
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal	NaN	NaN	NaN	NaN	0	/content/drive/MyDrive/Capstone/Pneumonia/Data...
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity	264.0	152.0	213.0	379.0	1	/content/drive/MyDrive/Capstone/Pneumonia/Data...

pneumonia shape

```
(37629, 8)
```

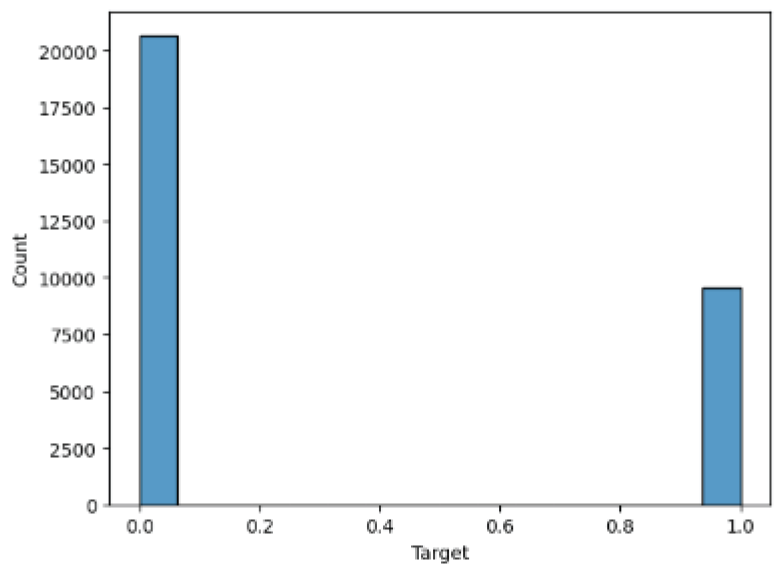
1) Looks like sum duplicate records come while merging, let's use unique record

Step 4. Preprocessing and Visualisations of different classes.

```
pneumonia_df.shape

(30227, 8)

pneumonia_df['patientId'].nunique()

26684
```

Let create Histrigram to plot the dataset

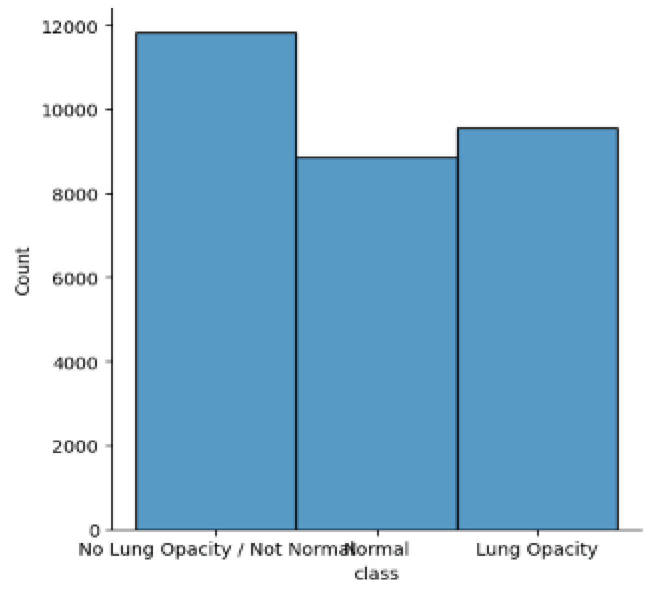
```
<Axes: xlabel='Target', ylabel='Count'>
```



1) We have an imbalance in the target data.
2) Class 0 - Normal, No Lung Opacity/Not Normal - 20672
3) Class 1 - Lung Opacity – 9555

Lets try with Distplot

```
<seaborn.axisgrid.FacetGrid at 0x784ed890ca90>
```

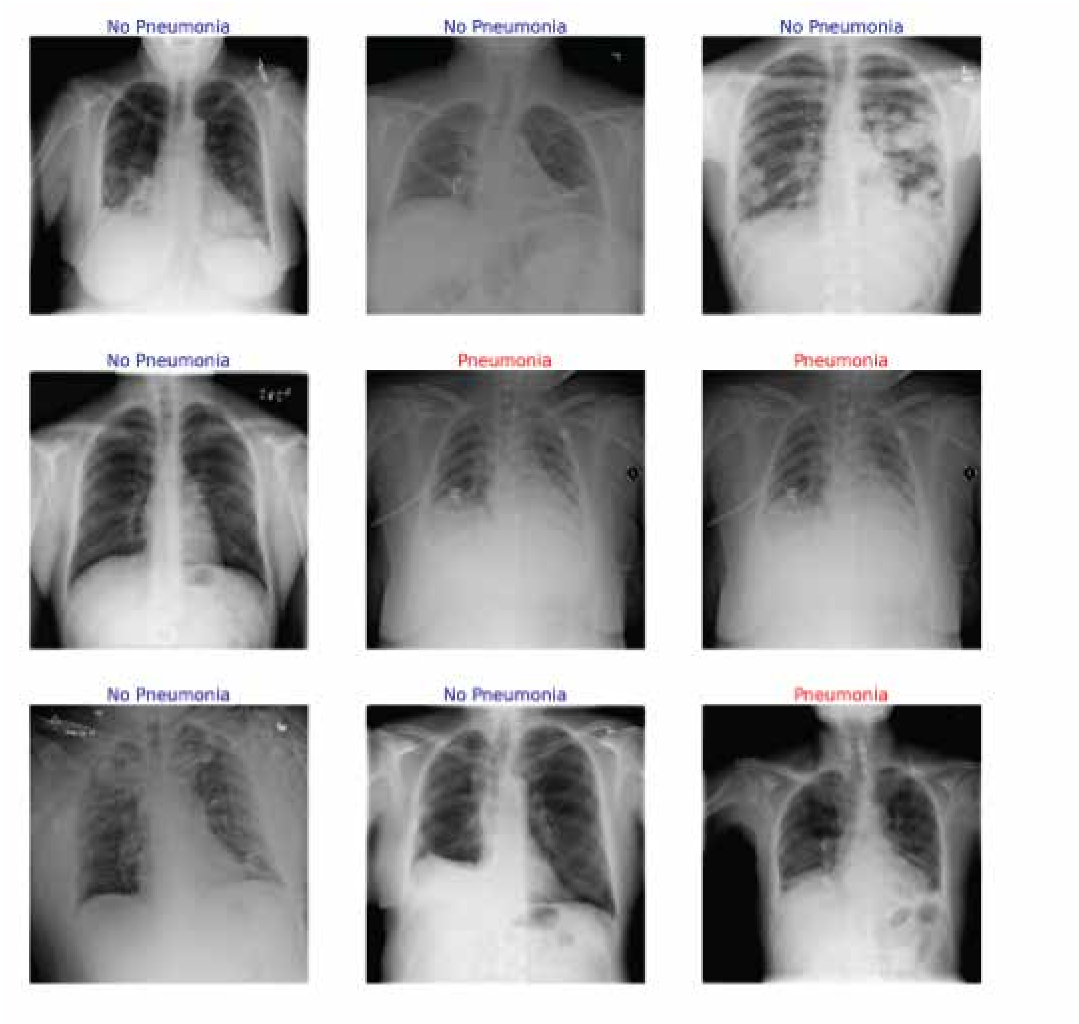


1) The above clearly shows that the dataset is imbalanced. We can use data augmentation techniques to make a balanced dataset.

#converting the image to a pixel values and resizing the array values. #Resize image to 224x224

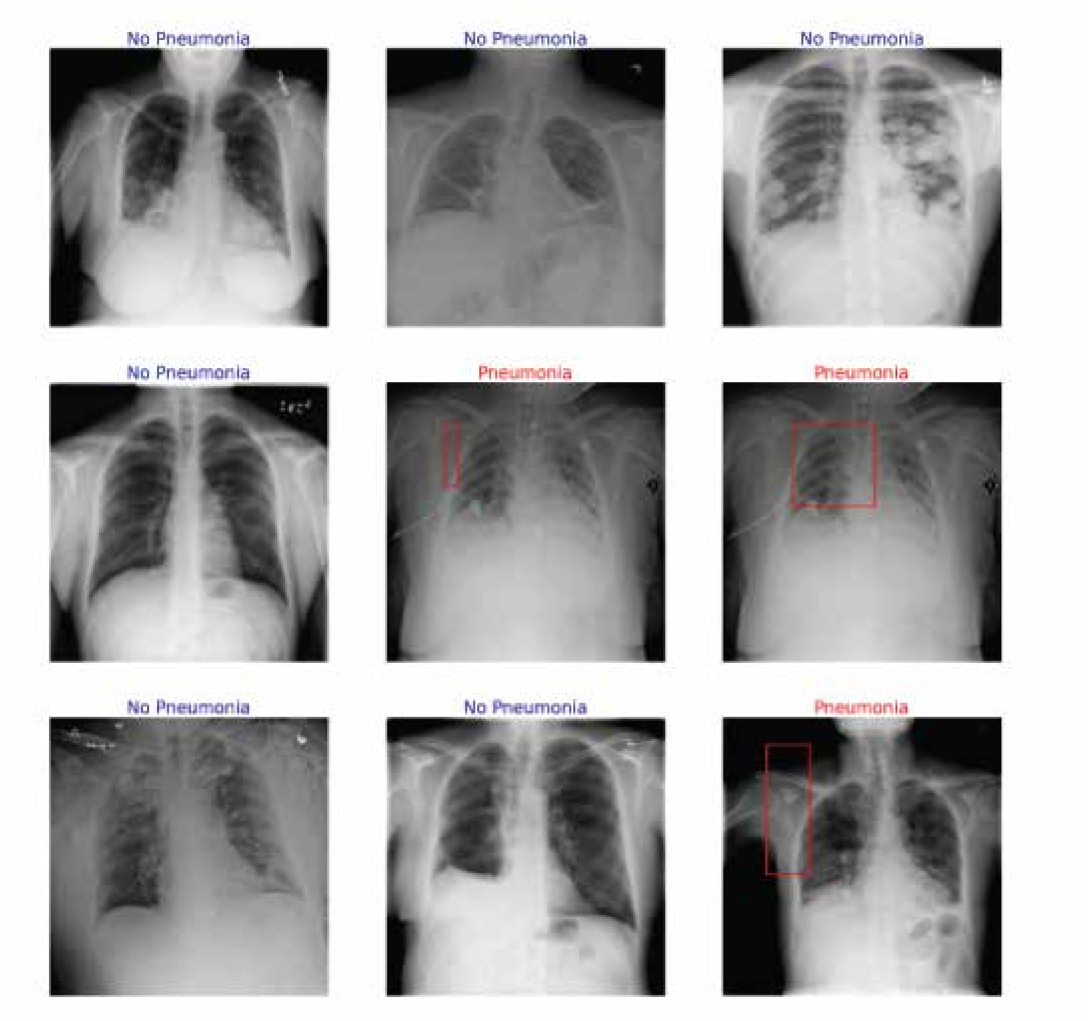
```
len(resized_images)

5000

resized_images[0].shape

(224, 224, 3)
```

#Plot 3*3 for Pneumonia Detection



1) As given in the input we can see blue color fonts for the No-Pneumonia & Red Color font for the Pneumonia.

Step 5. Display images with a bounding box.



1) As given in the input we can see the bonding Box for the Exact location where the Pneumonia is located for future process.

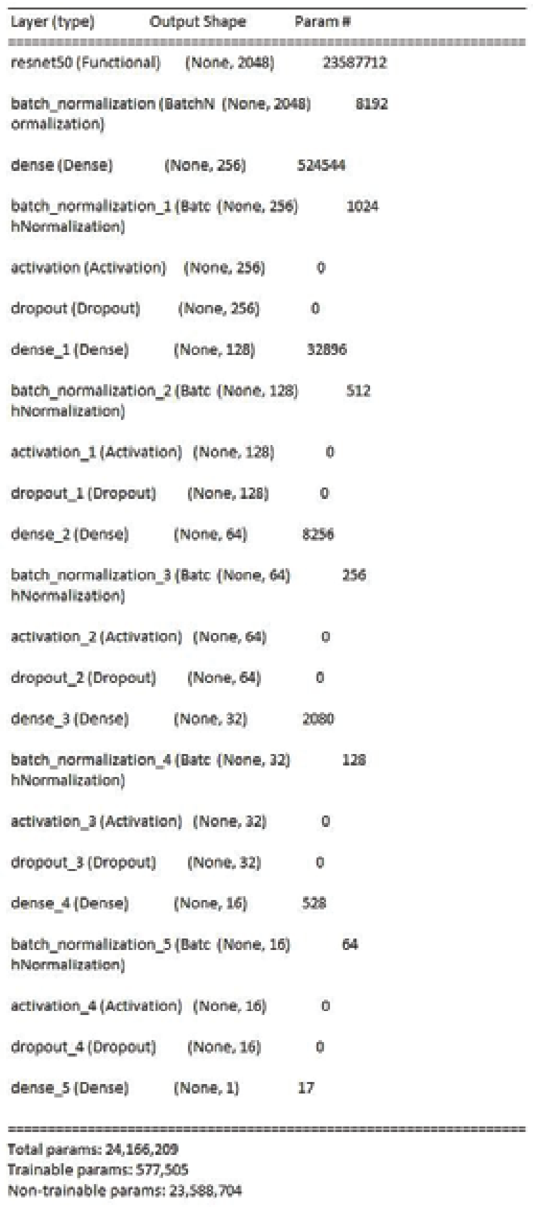
Data Preparation & Model Building

Step 6. Design, train and test basic CNN models for classification.

```
4000 4000
1000 1000
```

1) As given in the input we can see the train and test are data are balanced equally.

Using ResNet50 as a base model



```
Layer (type)            Output Shape          Param #
=================================================================
resnet50 (Functional)     (None, 2048)         23587712

batch_normalization (BatchN  (None, 2048)        8192
ormalization)

dense (Dense)            (None, 256)          524544

batch_normalization_1 (Batc  (None, 256)         1024
hNormalization)

activation (Activation)    (None, 256)          0

dropout (Dropout)         (None, 256)          0

dense_1 (Dense)           (None, 128)          32896

batch_normalization_2 (Batc  (None, 128)         512
hNormalization)

activation_1 (Activation)   (None, 128)          0

dropout_1 (Dropout)        (None, 128)          0

dense_2 (Dense)           (None, 64)           8256

batch_normalization_3 (Batc  (None, 64)          256
hNormalization)

activation_2 (Activation)   (None, 64)           0

dropout_2 (Dropout)        (None, 64)           0

dense_3 (Dense)           (None, 32)           2080

batch_normalization_4 (Batc  (None, 32)          128
hNormalization)

activation_3 (Activation)   (None, 32)           0

dropout_3 (Dropout)        (None, 32)           0

dense_4 (Dense)           (None, 16)           528

batch_normalization_5 (Batc  (None, 16)          64
hNormalization)

activation_4 (Activation)   (None, 16)           0

dropout_4 (Dropout)        (None, 16)           0

dense_5 (Dense)           (None, 1)            17

=================================================================
Total params: 24,166,209
Trainable params: 577,505
Non-trainable params: 23,588,704
```

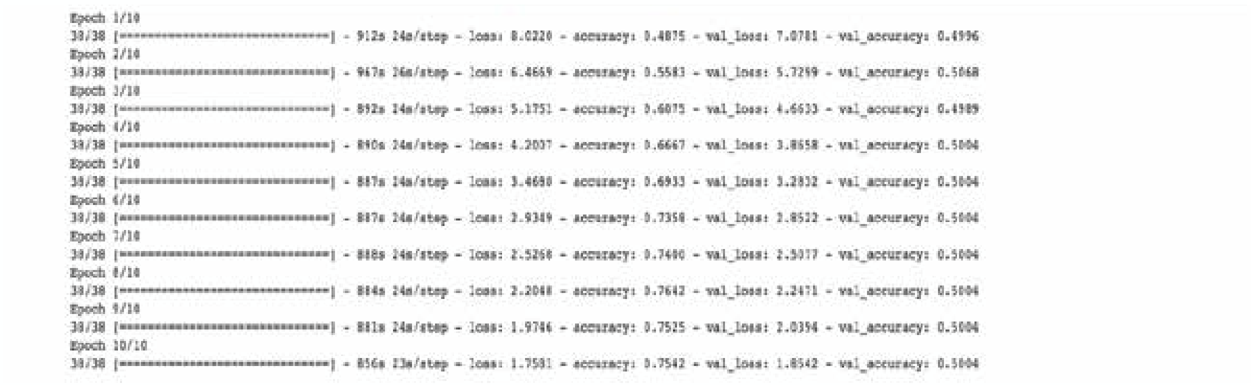
train shape

```
(4000, 224, 224, 3)
```

y train

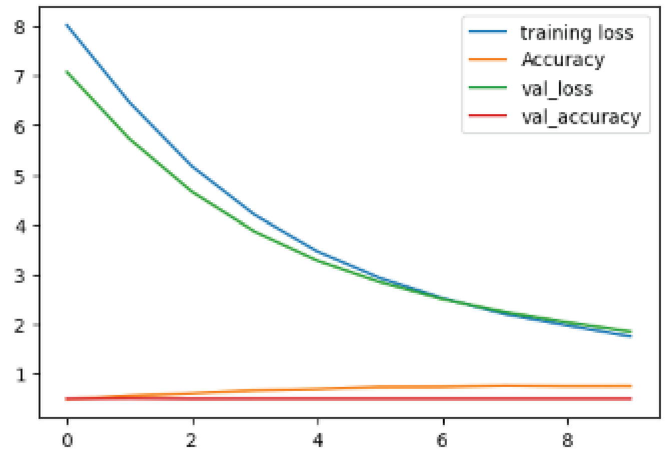
```
0.0
```

Model Compilation



```
Epoch 1/10
38/38 [==============================] - 912s 24s/step - loss: 8.0220 - accuracy: 0.4875 - val_loss: 7.0781 - val_accuracy: 0.4996
Epoch 2/10
38/38 [==============================] - 967s 26s/step - loss: 6.4669 - accuracy: 0.5583 - val_loss: 5.7299 - val_accuracy: 0.5068
Epoch 3/10
38/38 [==============================] - 892s 24s/step - loss: 5.1751 - accuracy: 0.6075 - val_loss: 4.6633 - val_accuracy: 0.4989
Epoch 4/10
38/38 [==============================] - 890s 24s/step - loss: 4.2037 - accuracy: 0.6667 - val_loss: 3.8658 - val_accuracy: 0.5004
Epoch 5/10
38/38 [==============================] - 887s 24s/step - loss: 3.4680 - accuracy: 0.6933 - val_loss: 3.2832 - val_accuracy: 0.5004
Epoch 6/10
38/38 [==============================] - 887s 24s/step - loss: 2.9349 - accuracy: 0.7358 - val_loss: 2.8522 - val_accuracy: 0.5004
Epoch 7/10
38/38 [==============================] - 888s 24s/step - loss: 2.5268 - accuracy: 0.7400 - val_loss: 2.5077 - val_accuracy: 0.5004
Epoch 8/10
38/38 [==============================] - 884s 24s/step - loss: 2.2048 - accuracy: 0.7642 - val_loss: 2.2471 - val_accuracy: 0.5004
Epoch 9/10
38/38 [==============================] - 881s 24s/step - loss: 1.9746 - accuracy: 0.7525 - val_loss: 2.0394 - val_accuracy: 0.5004
Epoch 10/10
38/38 [==============================] - 856s 23s/step - loss: 1.7581 - accuracy: 0.7542 - val_loss: 1.8542 - val_accuracy: 0.5004
```

Plot Accuracy and Loss Chart



Conclusion

Model Accuracy looks good on the sample data. We used only 5000 total records for this project because of the compute constraint. We tried with full data but google colab crashes.

So we decided as a team to go with 5000 samples, we split this into a train set of 4000 and a test set with 1000 samples.

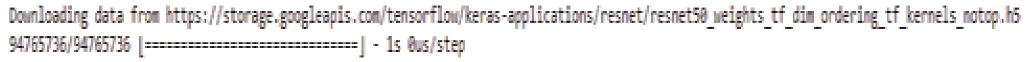
Model Accuracy is ~75%
Model Validation Accuracy is ~50%

Our rationale behind low validation accuracy is low sample size because of compute constraints. If we run this algorithm with sufficient compute power we will get good validation accuracy also.

Milestone 2

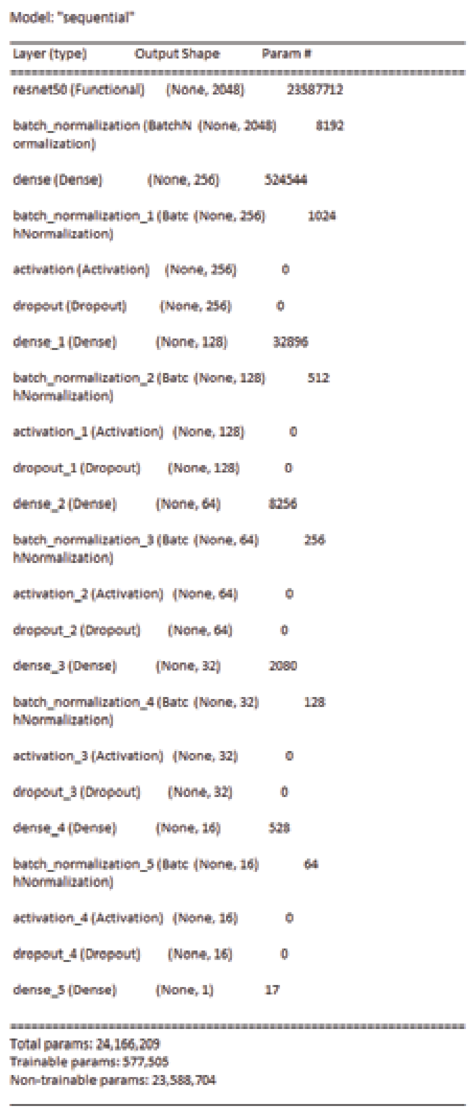
Fine tune the trained basic CNN models for classification.

# Using ResNet50 as a base model



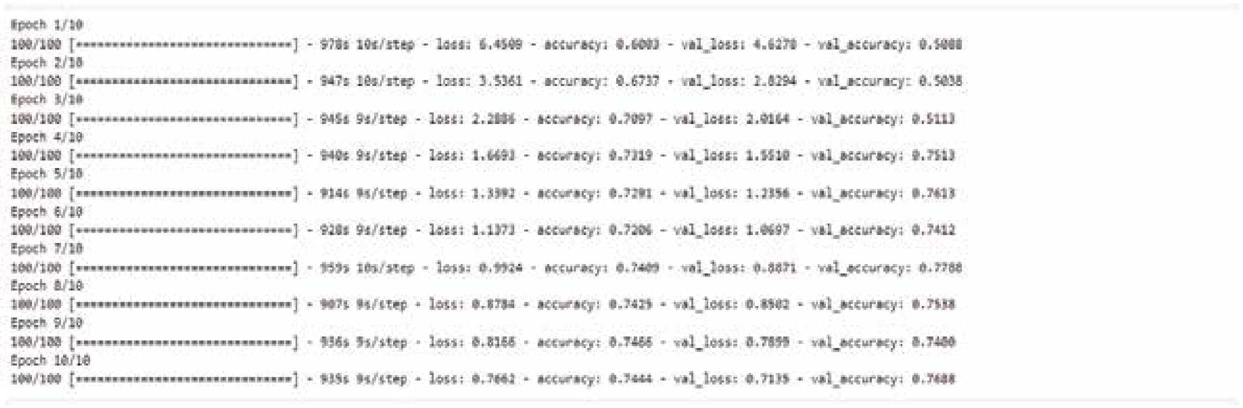
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
```

tuned_model_resnet.summary()



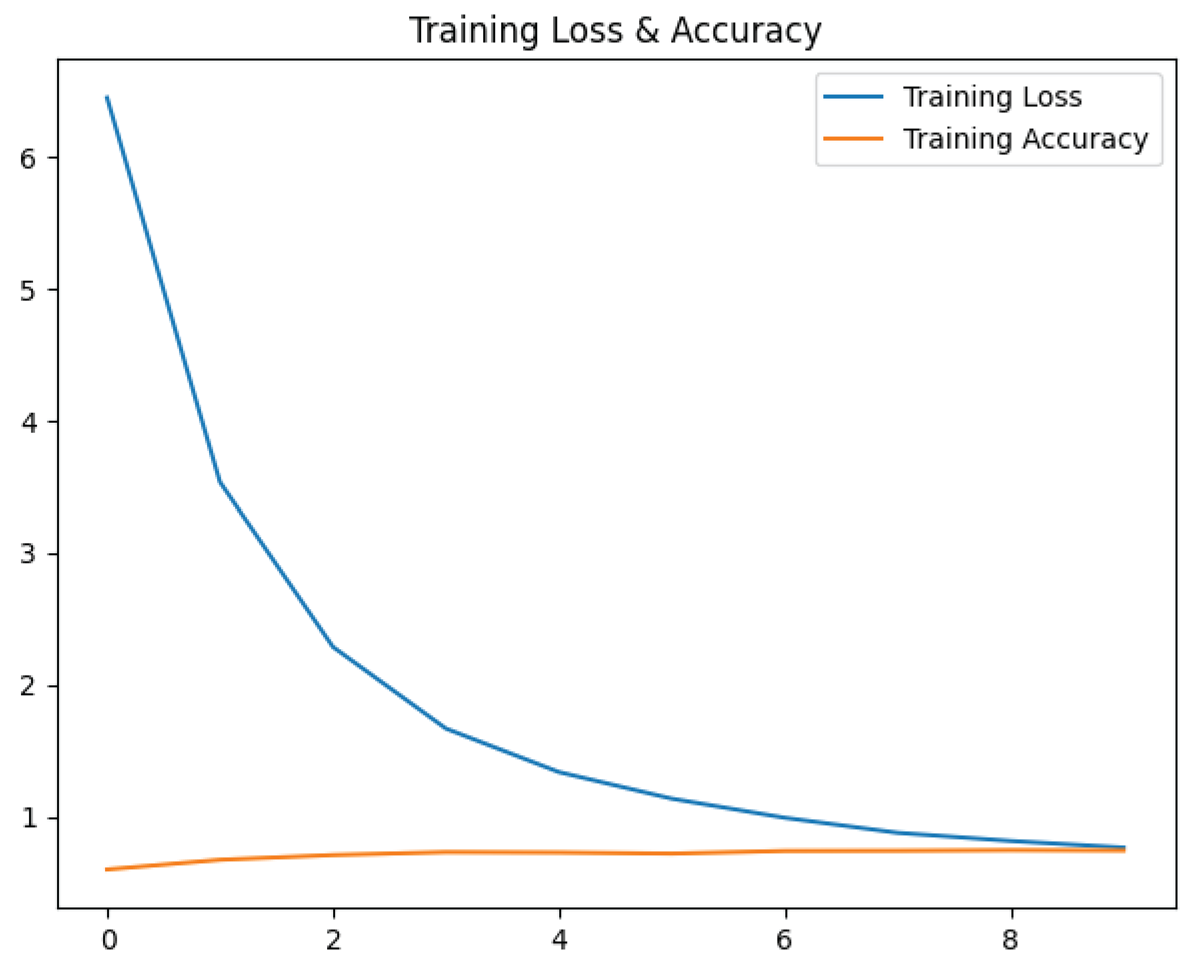
```
Model: "sequential"
_________________________________________________________________
 Layer (type)              Output Shape           Param #
=================================================================
 resnet50 (Functional)      (None, 2048)           23587712

 batch_normalization (BatchN  (None, 2048)          8192
 ormalization)

 dense (Dense)              (None, 256)            524544

 batch_normalization_1 (Batc  (None, 256)           1024
 hNormalization)

 activation (Activation)    (None, 256)            0

 dropout (Dropout)          (None, 256)            0

 dense_1 (Dense)            (None, 128)            32896

 batch_normalization_2 (Batc  (None, 128)           512
 hNormalization)

 activation_1 (Activation)  (None, 128)            0

 dropout_1 (Dropout)        (None, 128)            0

 dense_2 (Dense)            (None, 64)             8256

 batch_normalization_3 (Batc  (None, 64)            256
 hNormalization)

 activation_2 (Activation)  (None, 64)             0

 dropout_2 (Dropout)        (None, 64)             0

 dense_3 (Dense)            (None, 32)             2080

 batch_normalization_4 (Batc  (None, 32)            128
 hNormalization)

 activation_3 (Activation)  (None, 32)             0

 dropout_3 (Dropout)        (None, 32)             0

 dense_4 (Dense)            (None, 16)             528

 batch_normalization_5 (Batc  (None, 16)            64
 hNormalization)

 activation_4 (Activation)  (None, 16)             0

 dropout_4 (Dropout)        (None, 16)             0

 dense_5 (Dense)            (None, 1)              17

=================================================================
Total params: 24,166,209
Trainable params: 577,505
Non-trainable params: 23,588,704
_________________________________________________________________
```

#Training Model for 10 Epochs

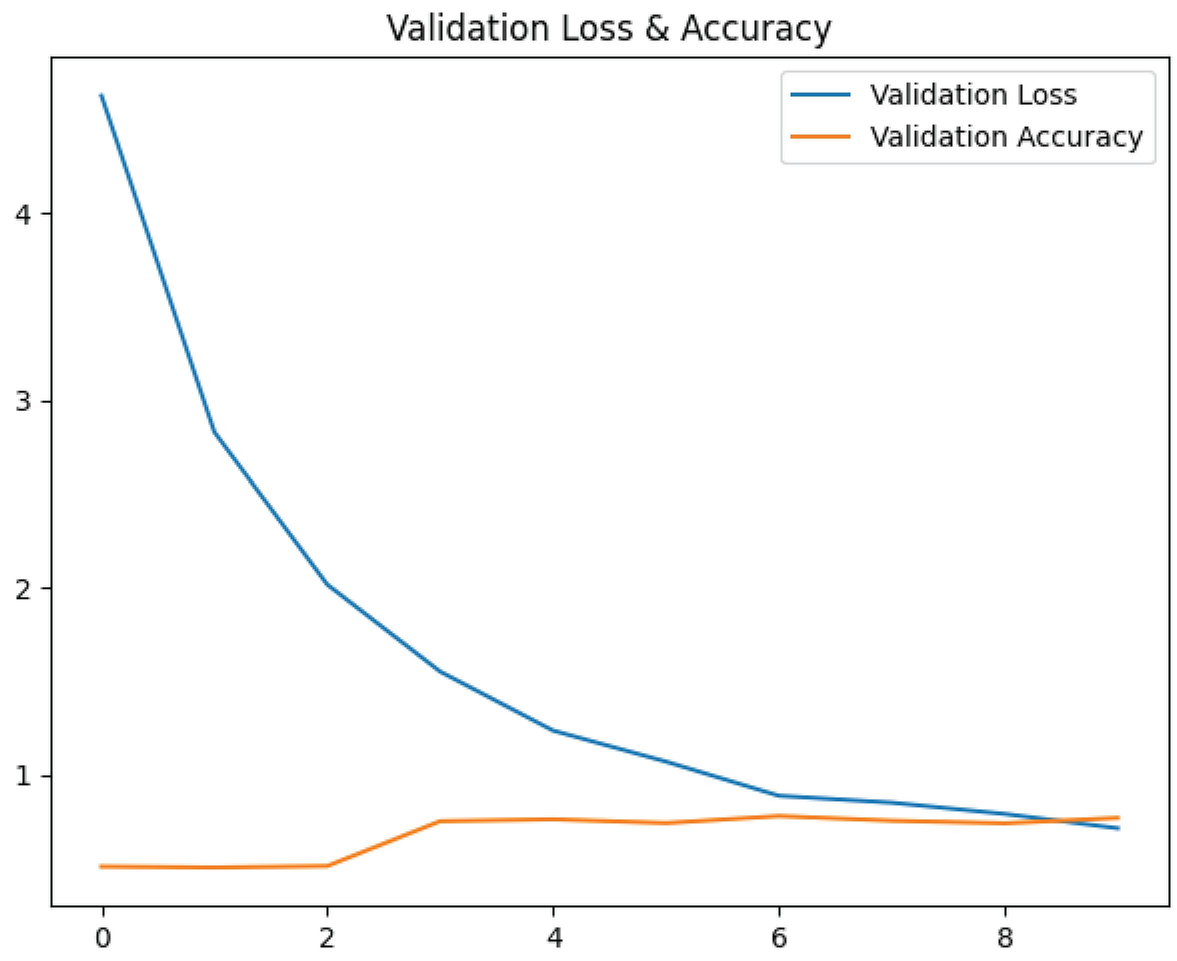


```
Epoch 1/10
100/100 [==============================] - 978s 10s/step - loss: 6.4509 - accuracy: 0.6003 - val_loss: 4.6270 - val_accuracy: 0.5088
Epoch 2/10
100/100 [==============================] - 947s 10s/step - loss: 3.5361 - accuracy: 0.6737 - val_loss: 2.8294 - val_accuracy: 0.5038
Epoch 3/10
100/100 [==============================] - 945s 9s/step - loss: 2.2886 - accuracy: 0.7097 - val_loss: 2.0164 - val_accuracy: 0.5113
Epoch 4/10
100/100 [==============================] - 940s 9s/step - loss: 1.6693 - accuracy: 0.7319 - val_loss: 1.5510 - val_accuracy: 0.7513
Epoch 5/10
100/100 [==============================] - 914s 9s/step - loss: 1.3392 - accuracy: 0.7291 - val_loss: 1.2356 - val_accuracy: 0.7613
Epoch 6/10
100/100 [==============================] - 928s 9s/step - loss: 1.1373 - accuracy: 0.7206 - val_loss: 1.0697 - val_accuracy: 0.7412
Epoch 7/10
100/100 [==============================] - 959s 10s/step - loss: 0.9924 - accuracy: 0.7409 - val_loss: 0.8871 - val_accuracy: 0.7788
Epoch 8/10
100/100 [==============================] - 907s 9s/step - loss: 0.8784 - accuracy: 0.7429 - val_loss: 0.8502 - val_accuracy: 0.7538
Epoch 9/10
100/100 [==============================] - 936s 9s/step - loss: 0.8166 - accuracy: 0.7466 - val_loss: 0.7899 - val_accuracy: 0.7400
Epoch 10/10
100/100 [==============================] - 935s 9s/step - loss: 0.7662 - accuracy: 0.7444 - val_loss: 0.7135 - val_accuracy: 0.7688
```

# Plot training loss and validation



# Plot training accuracy and validation accuracy



Model Accuracy is looks good on the sample data. We used only 5000 total rcords for this project because of the compute constraint. We tried with full data but google colab crashes.

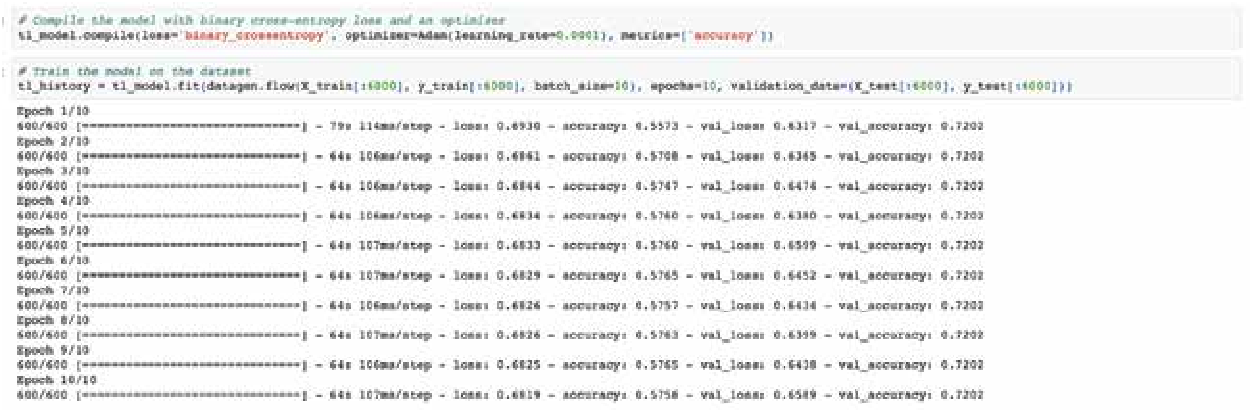
So we decided as team to go with 5000 samples, we split this into train set 4000 and test set with 1000 samples.

Model Accuracy is ~75%
Model Validation Accuracy is ~76%

After tuning the parameters of ResNet50, we can see evidence of significant improvement in Accuracy and Loss metrics.

# Trying EfficientNet

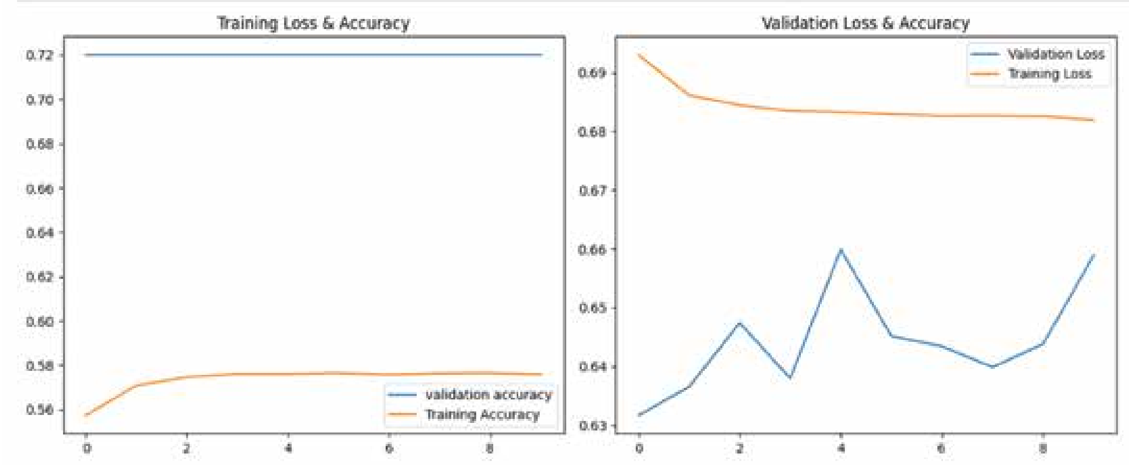
```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 0s 0us/step
Model: "model"
```



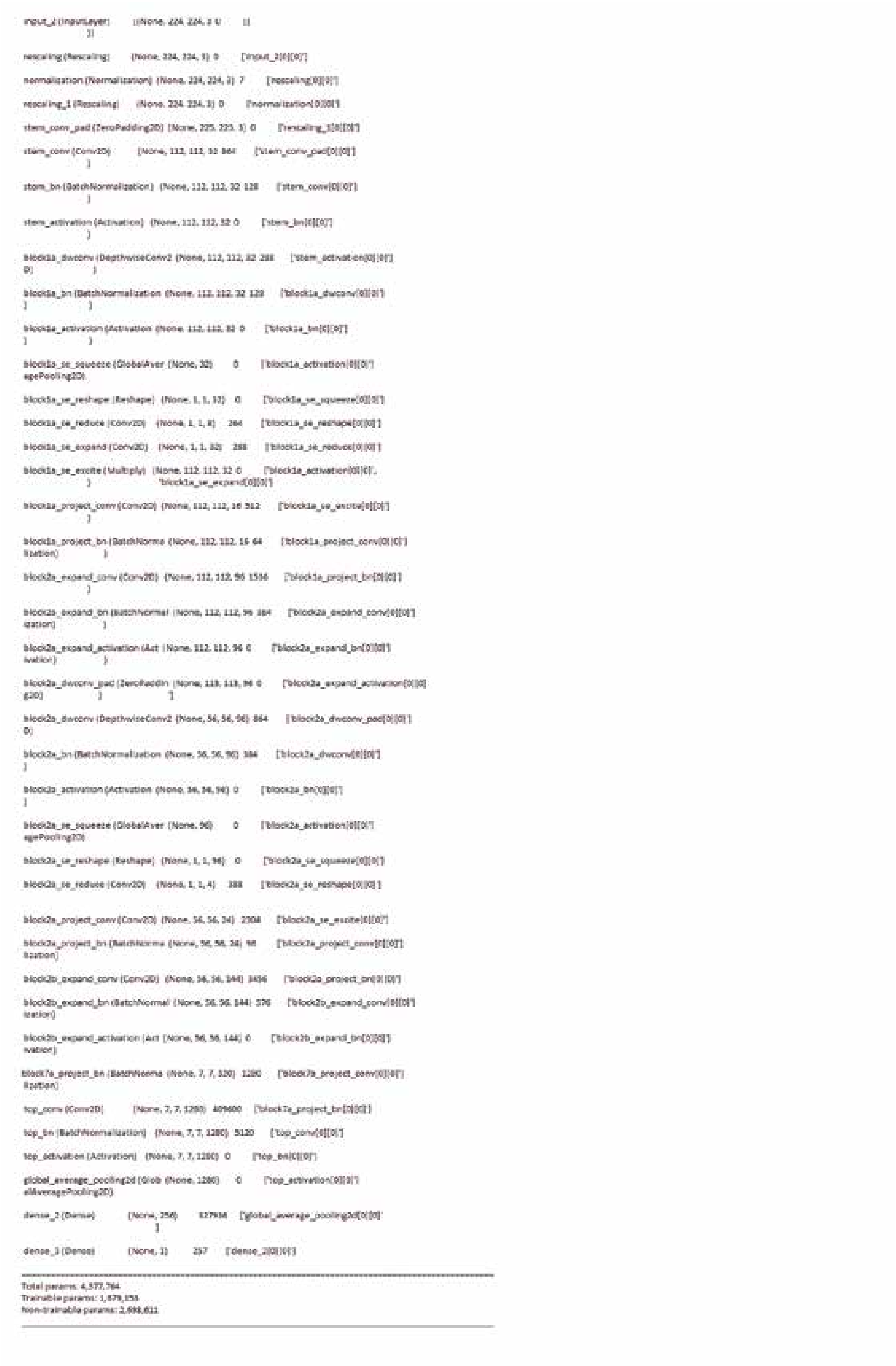
```
# Compile the model with binary cross-entropy loss and an optimizer
tl_model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001), metrics=['accuracy'])
```

```
# Train the model on the dataset
tl_history = tl_model.fit(datagen.flow(X_train[:6000], y_train[:6000], batch_size=10), epochs=10, validation_data=(X_test[:6000], y_test[:6000]))
```

```
Epoch 1/10
600/600 [==============================] - 79s 114ms/step - loss: 0.6930 - accuracy: 0.5573 - val_loss: 0.6317 - val_accuracy: 0.7202
Epoch 2/10
600/600 [==============================] - 64s 106ms/step - loss: 0.6861 - accuracy: 0.5708 - val_loss: 0.6365 - val_accuracy: 0.7202
Epoch 3/10
600/600 [==============================] - 64s 106ms/step - loss: 0.6844 - accuracy: 0.5747 - val_loss: 0.6474 - val_accuracy: 0.7202
Epoch 4/10
600/600 [==============================] - 64s 106ms/step - loss: 0.6834 - accuracy: 0.5760 - val_loss: 0.6380 - val_accuracy: 0.7202
Epoch 5/10
600/600 [==============================] - 64s 107ms/step - loss: 0.6833 - accuracy: 0.5760 - val_loss: 0.6599 - val_accuracy: 0.7202
Epoch 6/10
600/600 [==============================] - 64s 107ms/step - loss: 0.6829 - accuracy: 0.5765 - val_loss: 0.6452 - val_accuracy: 0.7202
Epoch 7/10
600/600 [==============================] - 64s 106ms/step - loss: 0.6826 - accuracy: 0.5757 - val_loss: 0.6434 - val_accuracy: 0.7202
Epoch 8/10
600/600 [==============================] - 64s 107ms/step - loss: 0.6826 - accuracy: 0.5763 - val_loss: 0.6399 - val_accuracy: 0.7202
Epoch 9/10
600/600 [==============================] - 64s 106ms/step - loss: 0.6825 - accuracy: 0.5765 - val_loss: 0.6438 - val_accuracy: 0.7202
Epoch 10/10
600/600 [==============================] - 64s 107ms/step - loss: 0.6819 - accuracy: 0.5758 - val_loss: 0.6589 - val_accuracy: 0.7202
```

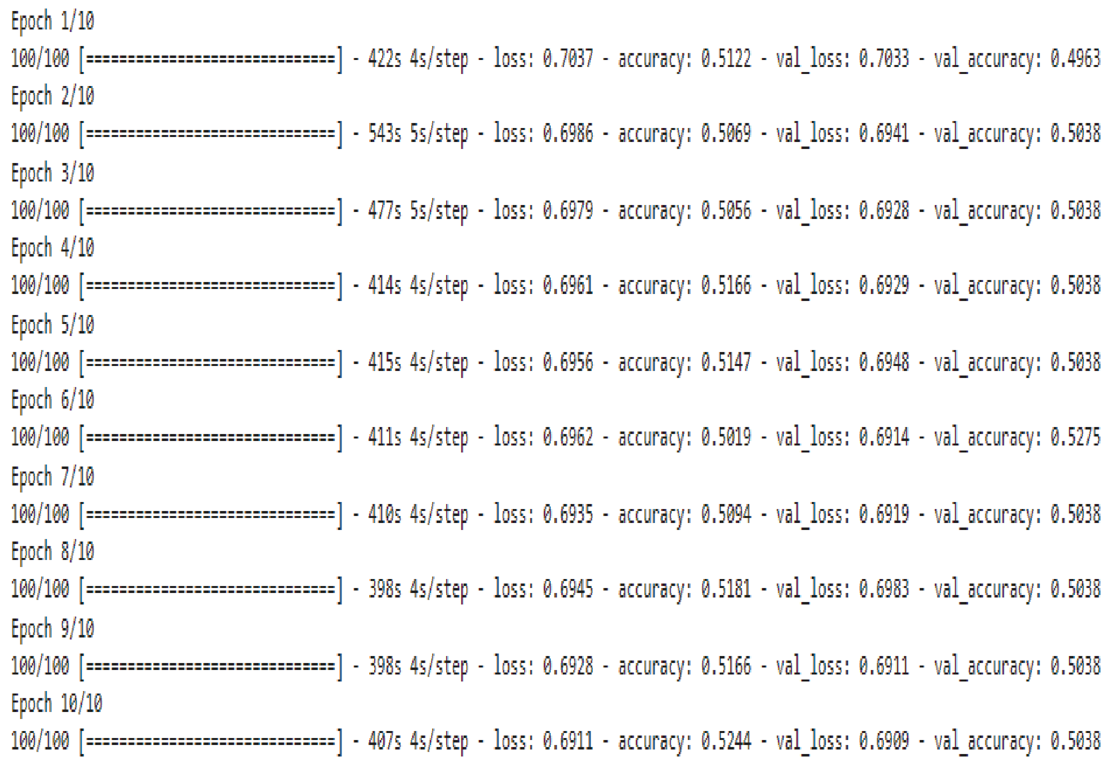


# Create the final model



```
input_2 (InputLayer)        [(None, 224, 224, 3 0        []
                  )]

rescaling (Rescaling)       (None, 224, 224, 3) 0        ['input_2[0][0]']

normalization (Normalization) (None, 224, 224, 3) 7       ['rescaling[0][0]']

rescaling_1 (Rescaling)     (None, 224, 224, 3) 0        ['normalization[0][0]']

stem_conv_pad (ZeroPadding2D) (None, 225, 225, 3) 0      ['rescaling_1[0][0]']

stem_conv (Conv2D)          (None, 112, 112, 32 864      ['stem_conv_pad[0][0]']
                  )

stem_bn (BatchNormalization) (None, 112, 112, 32 128      ['stem_conv[0][0]']
                  )

stem_activation (Activation) (None, 112, 112, 32 0        ['stem_bn[0][0]']
                  )

block1a_dwconv (DepthwiseConv2 (None, 112, 112, 32 288    ['stem_activation[0][0]']
D)                 )

block1a_bn (BatchNormalization (None, 112, 112, 32 128    ['block1a_dwconv[0][0]']
)                  )

block1a_activation (Activation (None, 112, 112, 32 0      ['block1a_bn[0][0]']
)                  )

block1a_se_squeeze (GlobalAver (None, 32)      0        ['block1a_activation[0][0]']
agePooling2D)

block1a_se_reshape (Reshape)  (None, 1, 1, 32)   0        ['block1a_se_squeeze[0][0]']

block1a_se_reduce (Conv2D)    (None, 1, 1, 8)    264      ['block1a_se_reshape[0][0]']

block1a_se_expand (Conv2D)    (None, 1, 1, 32)   288      ['block1a_se_reduce[0][0]']

block1a_se_excite (Multiply)  (None, 112, 112, 32 0       ['block1a_activation[0][0]',
                  )                     'block1a_se_expand[0][0]']

block1a_project_conv (Conv2D) (None, 112, 112, 16 512     ['block1a_se_excite[0][0]']
                  )

block1a_project_bn (BatchNorma (None, 112, 112, 16 64     ['block1a_project_conv[0][0]']
lization)           )

block2a_expand_conv (Conv2D)  (None, 112, 112, 96 1536    ['block1a_project_bn[0][0]']
                  )

block2a_expand_bn (BatchNormal (None, 112, 112, 96 384    ['block2a_expand_conv[0][0]']
ization)            )

block2a_expand_activation (Act (None, 112, 112, 96 0      ['block2a_expand_bn[0][0]']
ivation)            )

block2a_dwconv_pad (ZeroPaddin (None, 113, 113, 96 0      ['block2a_expand_activation[0][0]
g2D)                )                    ']

block2a_dwconv (DepthwiseConv2 (None, 56, 56, 96) 864     ['block2a_dwconv_pad[0][0]']
D)

block2a_bn (BatchNormalization (None, 56, 56, 96) 384     ['block2a_dwconv[0][0]']
)

block2a_activation (Activation (None, 56, 56, 96) 0       ['block2a_bn[0][0]']
)

block2a_se_squeeze (GlobalAver (None, 96)      0        ['block2a_activation[0][0]']
agePooling2D)

block2a_se_reshape (Reshape)  (None, 1, 1, 96)   0        ['block2a_se_squeeze[0][0]']

block2a_se_reduce (Conv2D)    (None, 1, 1, 4)    388      ['block2a_se_reshape[0][0]']


block2a_project_conv (Conv2D) (None, 56, 56, 24)  2304    ['block2a_se_excite[0][0]']

block2a_project_bn (BatchNorma (None, 56, 56, 24) 96      ['block2a_project_conv[0][0]']
lization)

block2b_expand_conv (Conv2D)  (None, 56, 56, 144) 3456    ['block2a_project_bn[0][0]']

block2b_expand_bn (BatchNormal (None, 56, 56, 144) 576    ['block2b_expand_conv[0][0]']
ization)

block2b_expand_activation (Act (None, 56, 56, 144) 0      ['block2b_expand_bn[0][0]']
ivation)

block7a_project_bn (BatchNorma (None, 7, 7, 320)  1280    ['block7a_project_conv[0][0]']
lization)

top_conv (Conv2D)           (None, 7, 7, 1280)  409600   ['block7a_project_bn[0][0]']

top_bn (BatchNormalization)  (None, 7, 7, 1280)  5120     ['top_conv[0][0]']

top_activation (Activation)  (None, 7, 7, 1280)  0        ['top_bn[0][0]']

global_average_pooling2d (Glob (None, 1280)     0        ['top_activation[0][0]']
alAveragePooling2D)

dense_2 (Dense)             (None, 256)      327936   ['global_average_pooling2d[0][0]'
                                   ]

dense_3 (Dense)             (None, 1)        257      ['dense_2[0][0]']

==================================================================================================
Total params: 4,377,764
Trainable params: 1,679,153
Non-trainable params: 2,698,611
__________________________________________________________________________________________________
```

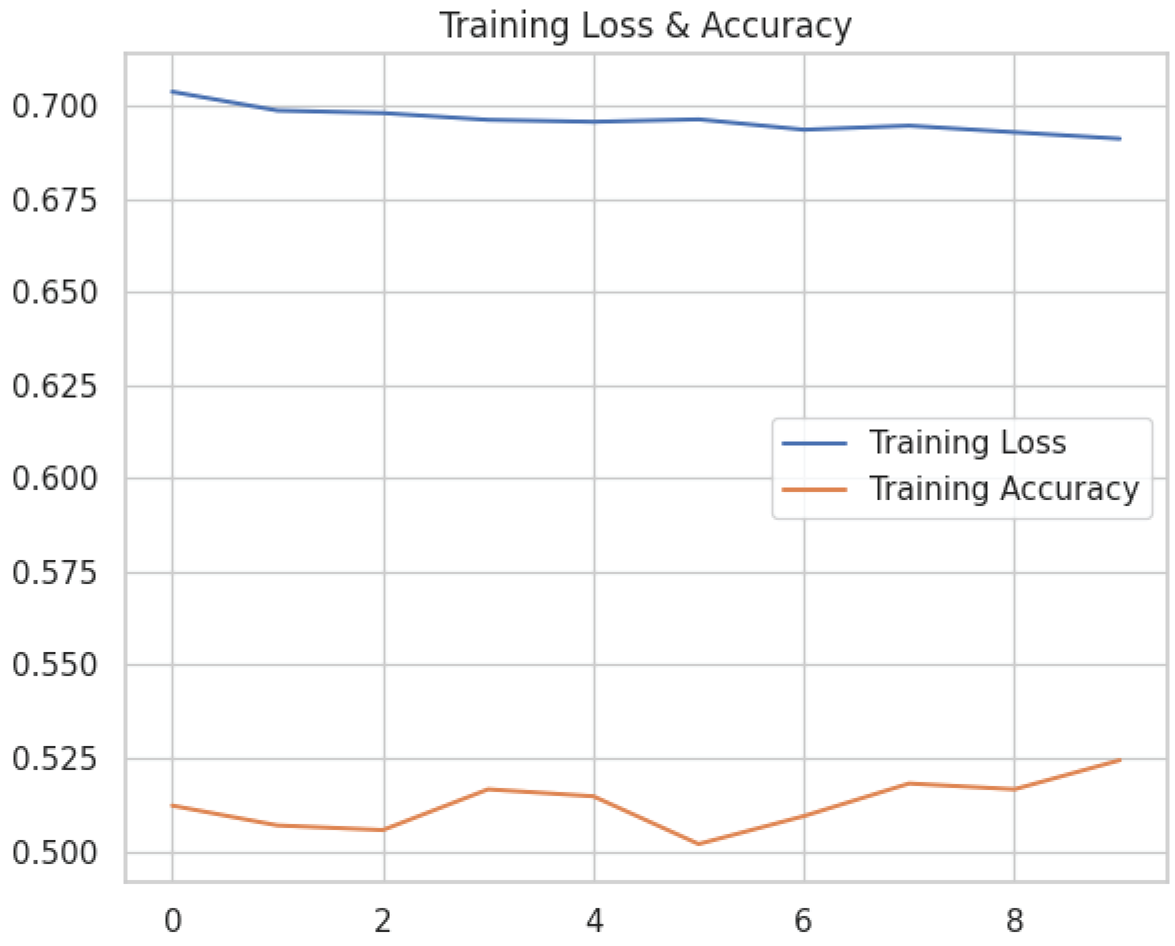
# Train the model on the dataset for 10 Epochs



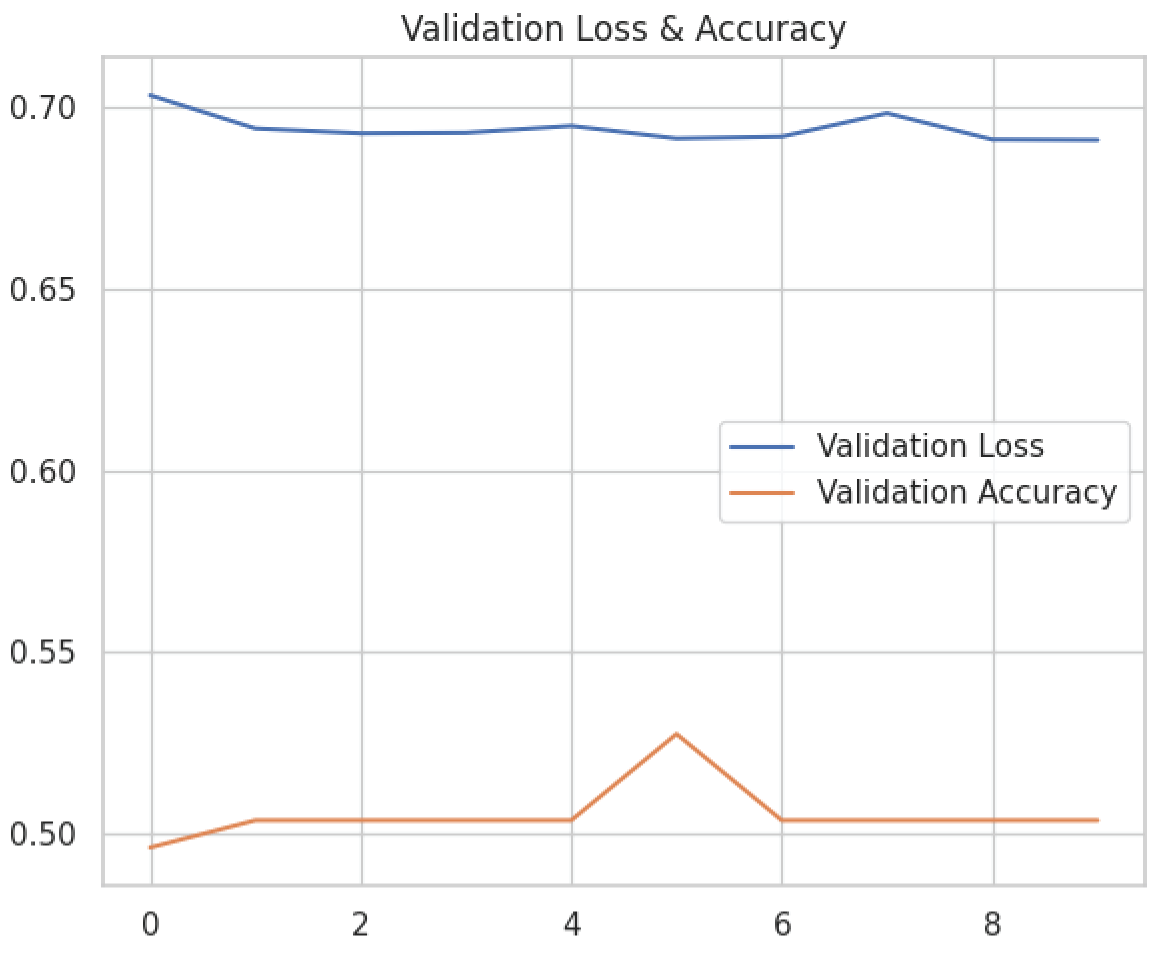
```
Epoch 1/10
100/100 [==============================] - 422s 4s/step - loss: 0.7037 - accuracy: 0.5122 - val_loss: 0.7033 - val_accuracy: 0.4963
Epoch 2/10
100/100 [==============================] - 543s 5s/step - loss: 0.6986 - accuracy: 0.5069 - val_loss: 0.6941 - val_accuracy: 0.5038
Epoch 3/10
100/100 [==============================] - 477s 5s/step - loss: 0.6979 - accuracy: 0.5056 - val_loss: 0.6928 - val_accuracy: 0.5038
Epoch 4/10
100/100 [==============================] - 414s 4s/step - loss: 0.6961 - accuracy: 0.5166 - val_loss: 0.6929 - val_accuracy: 0.5038
Epoch 5/10
100/100 [==============================] - 415s 4s/step - loss: 0.6956 - accuracy: 0.5147 - val_loss: 0.6948 - val_accuracy: 0.5038
Epoch 6/10
100/100 [==============================] - 411s 4s/step - loss: 0.6962 - accuracy: 0.5019 - val_loss: 0.6914 - val_accuracy: 0.5275
Epoch 7/10
100/100 [==============================] - 410s 4s/step - loss: 0.6935 - accuracy: 0.5094 - val_loss: 0.6919 - val_accuracy: 0.5038
Epoch 8/10
100/100 [==============================] - 398s 4s/step - loss: 0.6945 - accuracy: 0.5181 - val_loss: 0.6983 - val_accuracy: 0.5038
Epoch 9/10
100/100 [==============================] - 398s 4s/step - loss: 0.6928 - accuracy: 0.5166 - val_loss: 0.6911 - val_accuracy: 0.5038
Epoch 10/10
100/100 [==============================] - 407s 4s/step - loss: 0.6911 - accuracy: 0.5244 - val_loss: 0.6909 - val_accuracy: 0.5038
```

# Create a single figure with two subplots, arranged horizontally

# Plot training loss and validation loss

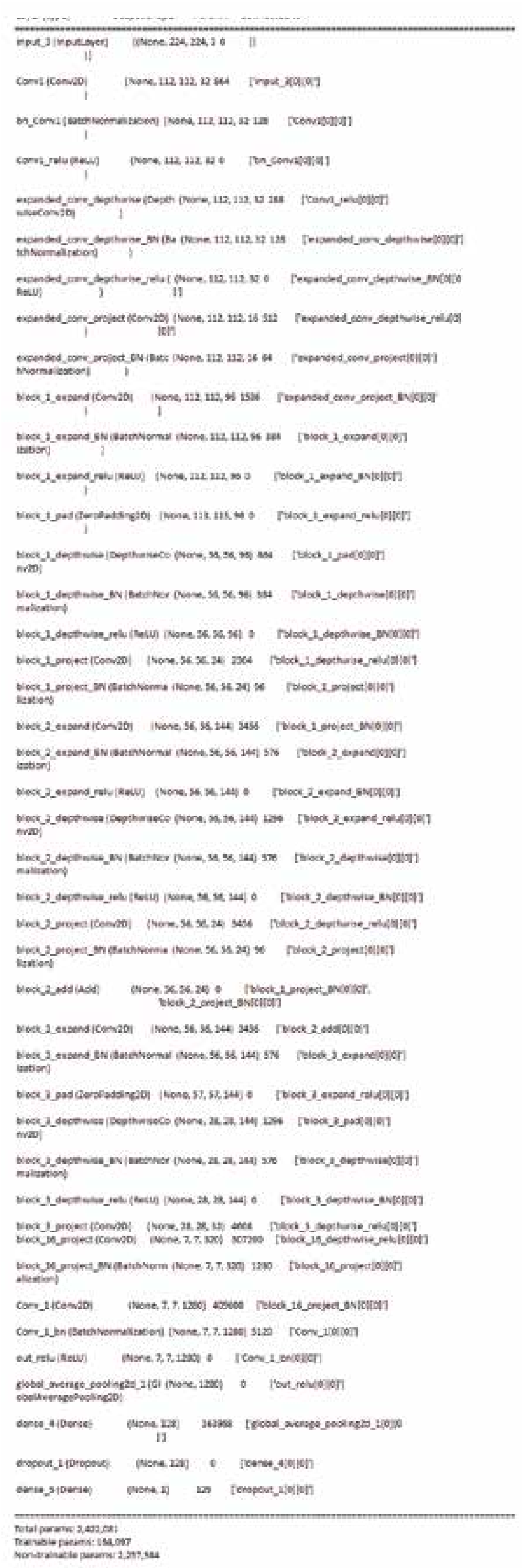


# Plot training accuracy and validation accuracy



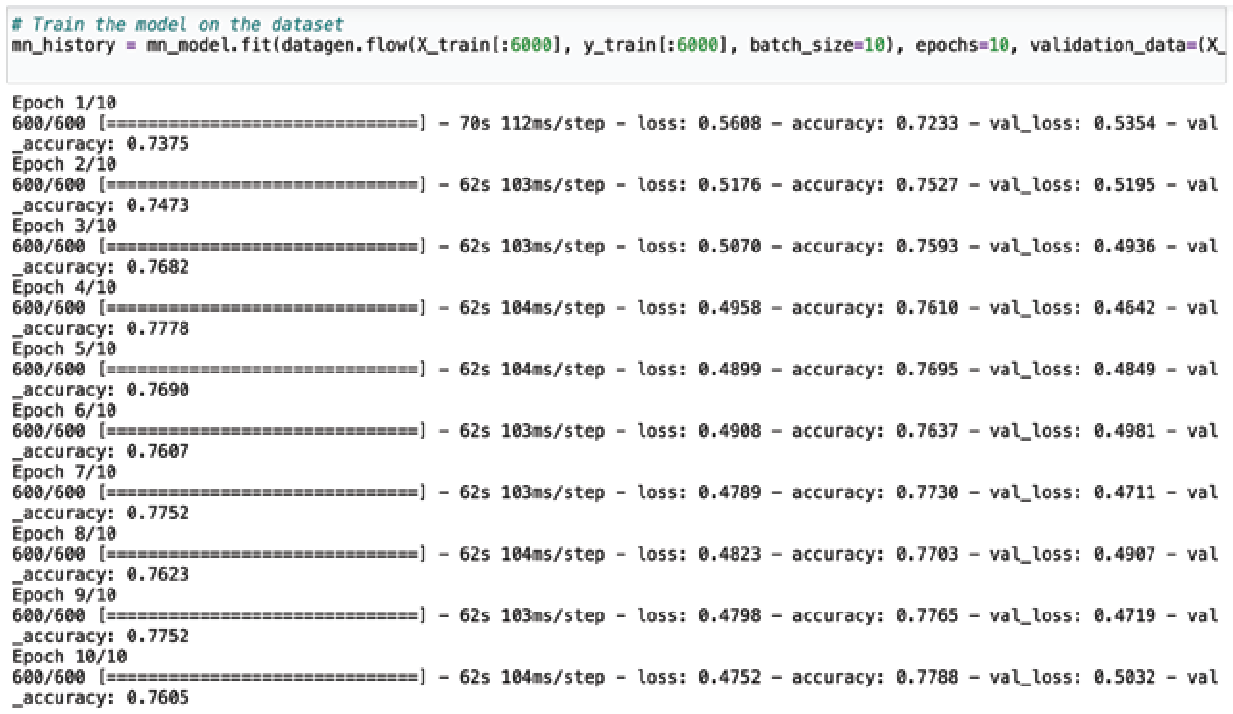
## MobileNetV2

# MobileNetV2 model pre-trained on ImageNet data



```
==================================================================================================
input_3 (InputLayer)       [(None, 224, 224, 3 0        []
                 )]

Conv1 (Conv2D)            (None, 112, 112, 32 864      ['input_3[0][0]']
                 )

bn_Conv1 (BatchNormalization) (None, 112, 112, 32 128     ['Conv1[0][0]']
                 )

Conv1_relu (ReLU)         (None, 112, 112, 32 0        ['bn_Conv1[0][0]']
                 )

expanded_conv_depthwise (Depth (None, 112, 112, 32 288     ['Conv1_relu[0][0]']
wiseConv2D)           )

expanded_conv_depthwise_BN (Ba (None, 112, 112, 32 128     ['expanded_conv_depthwise[0][0]']
tchNormalization)        )

expanded_conv_depthwise_relu ( (None, 112, 112, 32 0       ['expanded_conv_depthwise_BN[0][0
ReLU)              )                 ]']

expanded_conv_project (Conv2D) (None, 112, 112, 16 512     ['expanded_conv_depthwise_relu[0]
                 )                [0]']

expanded_conv_project_BN (Batc (None, 112, 112, 16 64      ['expanded_conv_project[0][0]']
hNormalization)          )

block_1_expand (Conv2D)     (None, 112, 112, 96 1536     ['expanded_conv_project_BN[0][0]'
                 )                ]

block_1_expand_BN (BatchNormal (None, 112, 112, 96 384     ['block_1_expand[0][0]']
ization)              )

block_1_expand_relu (ReLU)   (None, 112, 112, 96 0       ['block_1_expand_BN[0][0]']
                 )

block_1_pad (ZeroPadding2D)   (None, 113, 113, 96 0      ['block_1_expand_relu[0][0]']
                 )

block_1_depthwise (DepthwiseCo (None, 56, 56, 96) 864      ['block_1_pad[0][0]']
nv2D)

block_1_depthwise_BN (BatchNor (None, 56, 56, 96) 384      ['block_1_depthwise[0][0]']
malization)

block_1_depthwise_relu (ReLU) (None, 56, 56, 96) 0       ['block_1_depthwise_BN[0][0]']

block_1_project (Conv2D)     (None, 56, 56, 24) 2304     ['block_1_depthwise_relu[0][0]']

block_1_project_BN (BatchNorma (None, 56, 56, 24) 96      ['block_1_project[0][0]']
lization)

block_2_expand (Conv2D)     (None, 56, 56, 144) 3456     ['block_1_project_BN[0][0]']

block_2_expand_BN (BatchNormal (None, 56, 56, 144) 576     ['block_2_expand[0][0]']
ization)

block_2_expand_relu (ReLU)   (None, 56, 56, 144) 0      ['block_2_expand_BN[0][0]']

block_2_depthwise (DepthwiseCo (None, 56, 56, 144) 1296    ['block_2_expand_relu[0][0]']
nv2D)

block_2_depthwise_BN (BatchNor (None, 56, 56, 144) 576     ['block_2_depthwise[0][0]']
malization)

block_2_depthwise_relu (ReLU) (None, 56, 56, 144) 0      ['block_2_depthwise_BN[0][0]']

block_2_project (Conv2D)     (None, 56, 56, 24) 3456     ['block_2_depthwise_relu[0][0]']

block_2_project_BN (BatchNorma (None, 56, 56, 24) 96      ['block_2_project[0][0]']
lization)

block_2_add (Add)          (None, 56, 56, 24) 0       ['block_1_project_BN[0][0]',
                        'block_2_project_BN[0][0]']

block_3_expand (Conv2D)     (None, 56, 56, 144) 3456     ['block_2_add[0][0]']

block_3_expand_BN (BatchNormal (None, 56, 56, 144) 576     ['block_3_expand[0][0]']
ization)

block_3_pad (ZeroPadding2D)   (None, 57, 57, 144) 0      ['block_3_expand_relu[0][0]']

block_3_depthwise (DepthwiseCo (None, 28, 28, 144) 1296    ['block_3_pad[0][0]']
nv2D)

block_3_depthwise_BN (BatchNor (None, 28, 28, 144) 576     ['block_3_depthwise[0][0]']
malization)

block_3_depthwise_relu (ReLU) (None, 28, 28, 144) 0      ['block_3_depthwise_BN[0][0]']

block_3_project (Conv2D)     (None, 28, 28, 32) 4608     ['block_3_depthwise_relu[0][0]']
block_16_project (Conv2D)    (None, 7, 7, 320)  307200   ['block_16_depthwise_relu[0][0]']

block_16_project_BN (BatchNorm (None, 7, 7, 320)  1280    ['block_16_project[0][0]']
alization)

Conv_1 (Conv2D)           (None, 7, 7, 1280)  409600   ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalization) (None, 7, 7, 1280)  5120    ['Conv_1[0][0]']

out_relu (ReLU)           (None, 7, 7, 1280)  0       ['Conv_1_bn[0][0]']

global_average_pooling2d_1 (Gl (None, 1280)     0       ['out_relu[0][0]']
obalAveragePooling2D)

dense_4 (Dense)           (None, 128)      163968   ['global_average_pooling2d_1[0][0
                        ]']

dropout_1 (Dropout)         (None, 128)      0       ['dense_4[0][0]']

dense_5 (Dense)           (None, 1)       129     ['dropout_1[0][0]']

==================================================================================================
Total params: 2,422,081
Trainable params: 164,097
Non-trainable params: 2,257,984
```

Train the model on the dataset for 10 Epochs

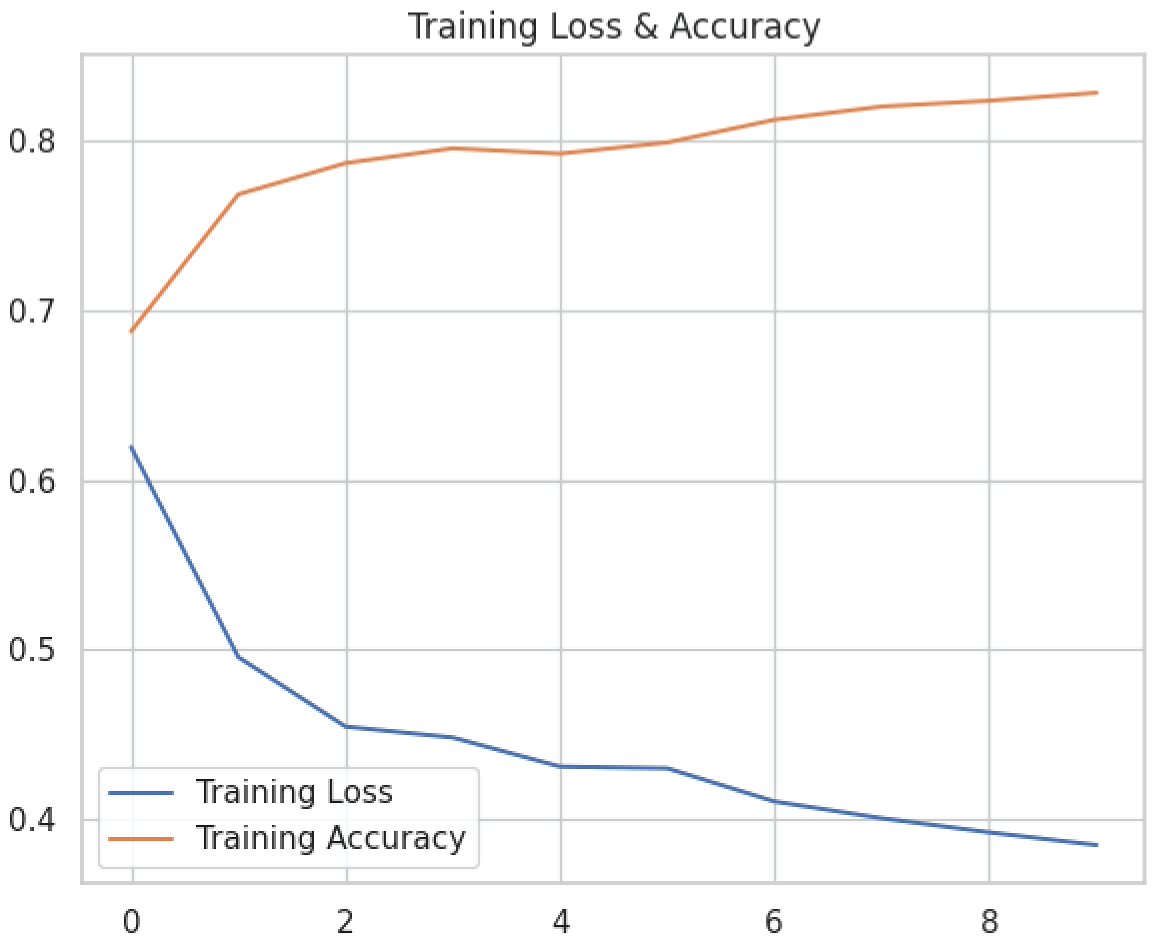


```
# Train the model on the dataset
mn_history = mn_model.fit(datagen.flow(X_train[:6000], y_train[:6000], batch_size=10), epochs=10, validation_data=(X_
```

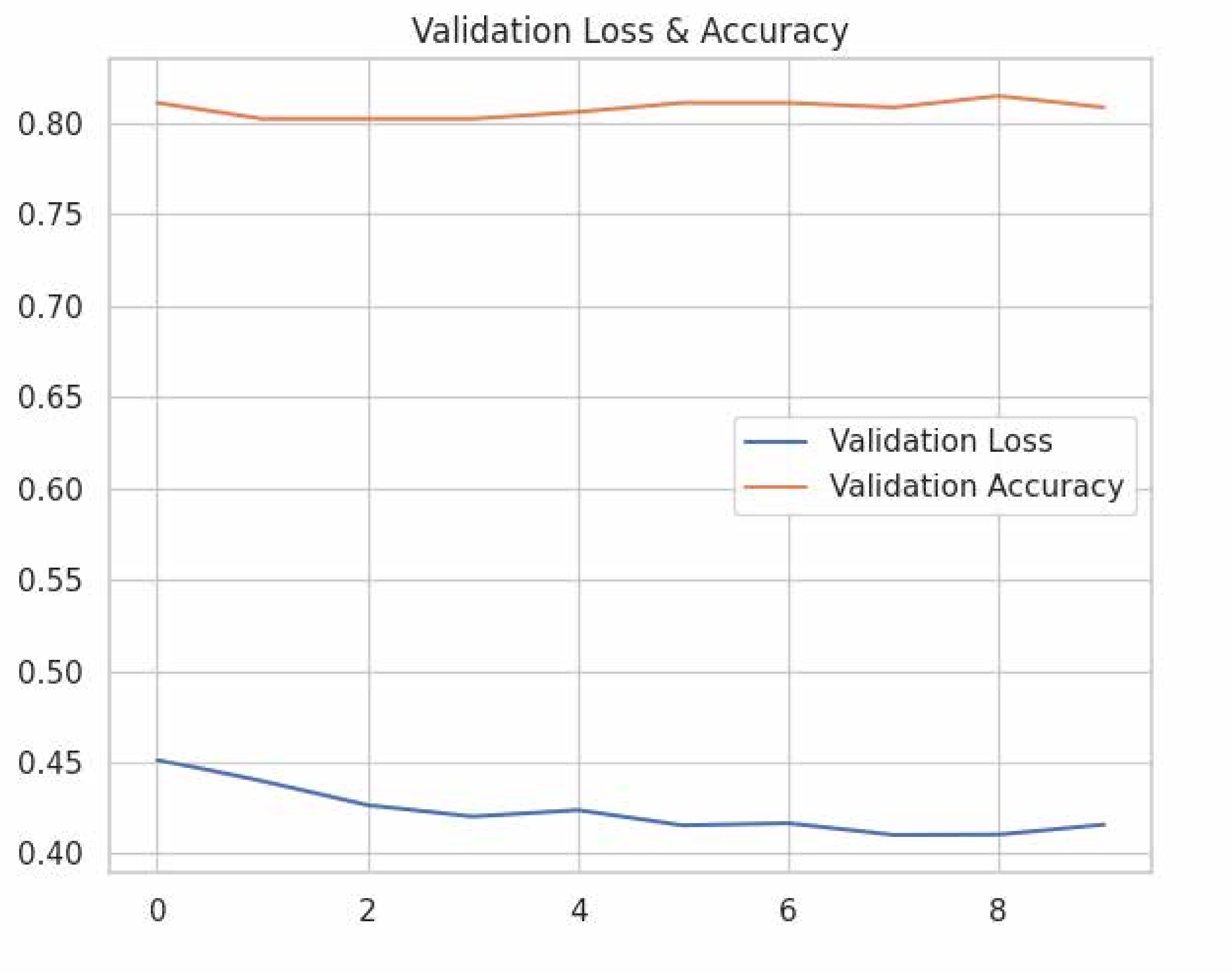
```
Epoch 1/10
600/600 [==============================] - 70s 112ms/step - loss: 0.5608 - accuracy: 0.7233 - val_loss: 0.5354 - val
_accuracy: 0.7375
Epoch 2/10
600/600 [==============================] - 62s 103ms/step - loss: 0.5176 - accuracy: 0.7527 - val_loss: 0.5195 - val
_accuracy: 0.7473
Epoch 3/10
600/600 [==============================] - 62s 103ms/step - loss: 0.5070 - accuracy: 0.7593 - val_loss: 0.4936 - val
_accuracy: 0.7682
Epoch 4/10
600/600 [==============================] - 62s 104ms/step - loss: 0.4958 - accuracy: 0.7610 - val_loss: 0.4642 - val
_accuracy: 0.7778
Epoch 5/10
600/600 [==============================] - 62s 104ms/step - loss: 0.4899 - accuracy: 0.7695 - val_loss: 0.4849 - val
_accuracy: 0.7690
Epoch 6/10
600/600 [==============================] - 62s 103ms/step - loss: 0.4908 - accuracy: 0.7637 - val_loss: 0.4981 - val
_accuracy: 0.7607
Epoch 7/10
600/600 [==============================] - 62s 103ms/step - loss: 0.4789 - accuracy: 0.7730 - val_loss: 0.4711 - val
_accuracy: 0.7752
Epoch 8/10
600/600 [==============================] - 62s 104ms/step - loss: 0.4823 - accuracy: 0.7703 - val_loss: 0.4907 - val
_accuracy: 0.7623
Epoch 9/10
600/600 [==============================] - 62s 103ms/step - loss: 0.4798 - accuracy: 0.7765 - val_loss: 0.4719 - val
_accuracy: 0.7752
Epoch 10/10
600/600 [==============================] - 62s 104ms/step - loss: 0.4752 - accuracy: 0.7788 - val_loss: 0.5032 - val
_accuracy: 0.7605
```

# Create a single figure with two subplots, arranged horizontally

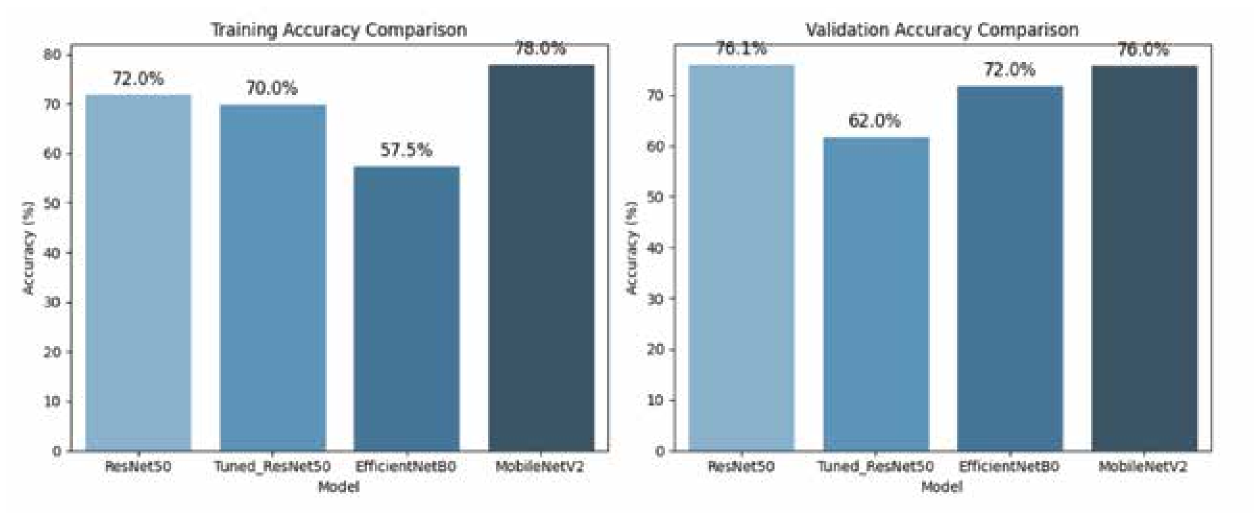
# Plot training loss and validation loss



# Plot training accuracy and validation accuracy



Adding Comparison bar chart among all CNN Models variations which we have used in this project.

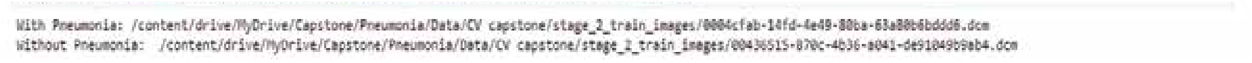


We can see clear evidence from the comparison chart, MobileNetV2 model is giving us very good results. Will evaluate MobileNetV2 with unknown data.

Evaluate MobileNetV2

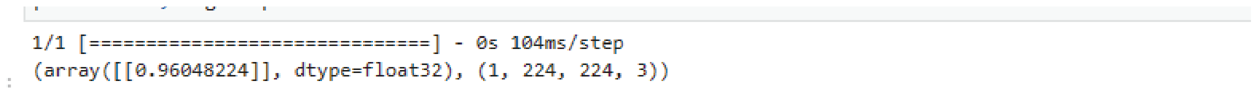
Evaluating MobileNetV2 model with unknown Image to check how it is reacting with new or unknown images.

# unknown sample images for Pneumonia detection



```
With Pneumonia: /content/drive/MyDrive/Capstone/Pneumonia/Data/CV capstone/stage_2_train_images/0004cfab-14fd-4e49-80ba-63a80b6bddd6.dcm
Without Pneumonia:  /content/drive/MyDrive/Capstone/Pneumonia/Data/CV capstone/stage_2_train_images/00436515-870c-4b36-a041-de91049b9ab4.dcm
```

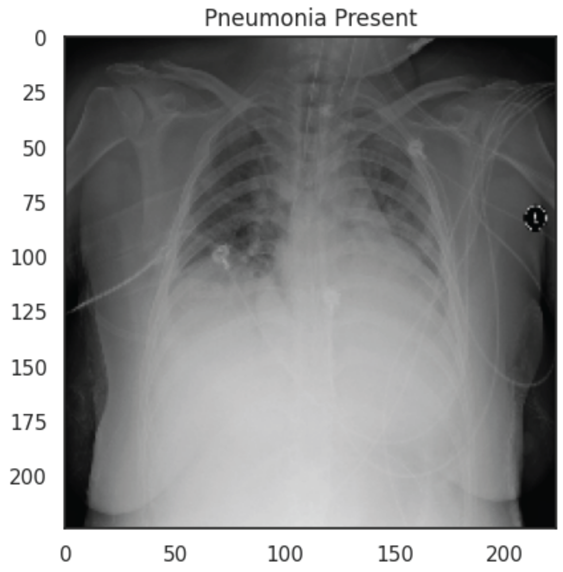
# Load and preprocess the unknown image



```
1/1 [==============================] - 0s 104ms/step
(array([[0.96048224]], dtype=float32), (1, 224, 224, 3))
```

# predicted probability

```
Predicted class: Pneumonia Present
Confidence: 0.9604822397232056
<matplotlib.image.AxesImage at 0x7bfd132ee770>
```



Model Accuracy looks good on the sample data. We used only 30000 total records for this project because of the compute constraint. We tried with full data but google colab crashes.

So we decided as a team to go with 30000 samples, we split this into a train set of 20000 and a test set with 10000 samples.

Model Accuracy of MobileNetV2 is ~78%
Model Validation Accuracy of MobileNetV2 is ~76%

MobileNetV2 model looks more appealing to me. It is giving more convincing results. Which accuracy is more close to the existing solution of pneumonia detection.

I have used a couple of sample unknown images with Pneumonia or without Pneumonia to check if our model is giving us the correct detection result or not. I am getting good results here too.

1. **With Pneumonia:** /content/drive/MyDrive/Capstone/Pneumonia/Data/CV capstone/stage_2_train_images/0004cfab-14fd-4e49-80ba-63a80b6bddd6.dcm
2. **Without Pneumonia:** /content/drive/MyDrive/Capstone/Pneumonia/Data/CV capstone/stage_2_train_images/00436515-870c-4b36-a041-de91049b9ab4.dcm

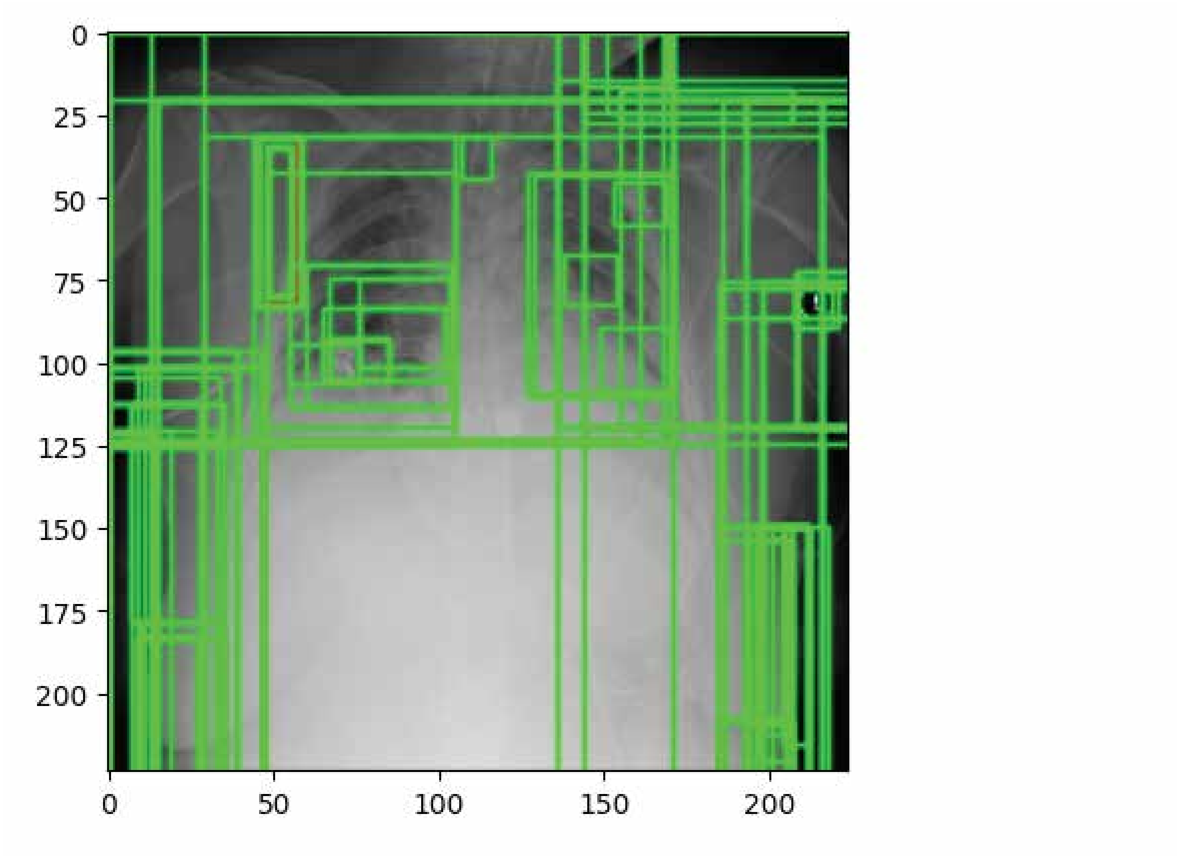
See results of the two above cells, we can just comment and uncomment the lines to check the detection results.

Step 3. Design, train and test RCNN & its hybrid based object detection models to impose the bounding box or mask over the area of interest.

The first step in RCNN is Selective search. Let's initialize Selective search using createSelectiveSearchSegmentation() class of opencv library.

Set one image as the base for selective search using setBaseImage(image)

Selective search segmentation function uses hierarchical clustering to group pixels and then combine them into one based on color, texture or composition. The following is an implementation of the search code on the base image.



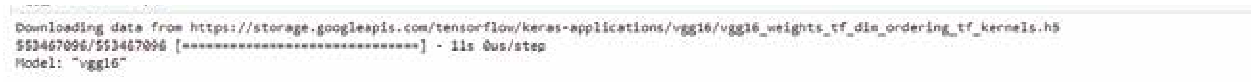
Loop over the image folder and set each image one by one as the base for selective search using setBaseImage(image) and get the proposed regions.

Initializing fast selective search and getting proposed regions using class switchToSelectiveSearchFast() and process().

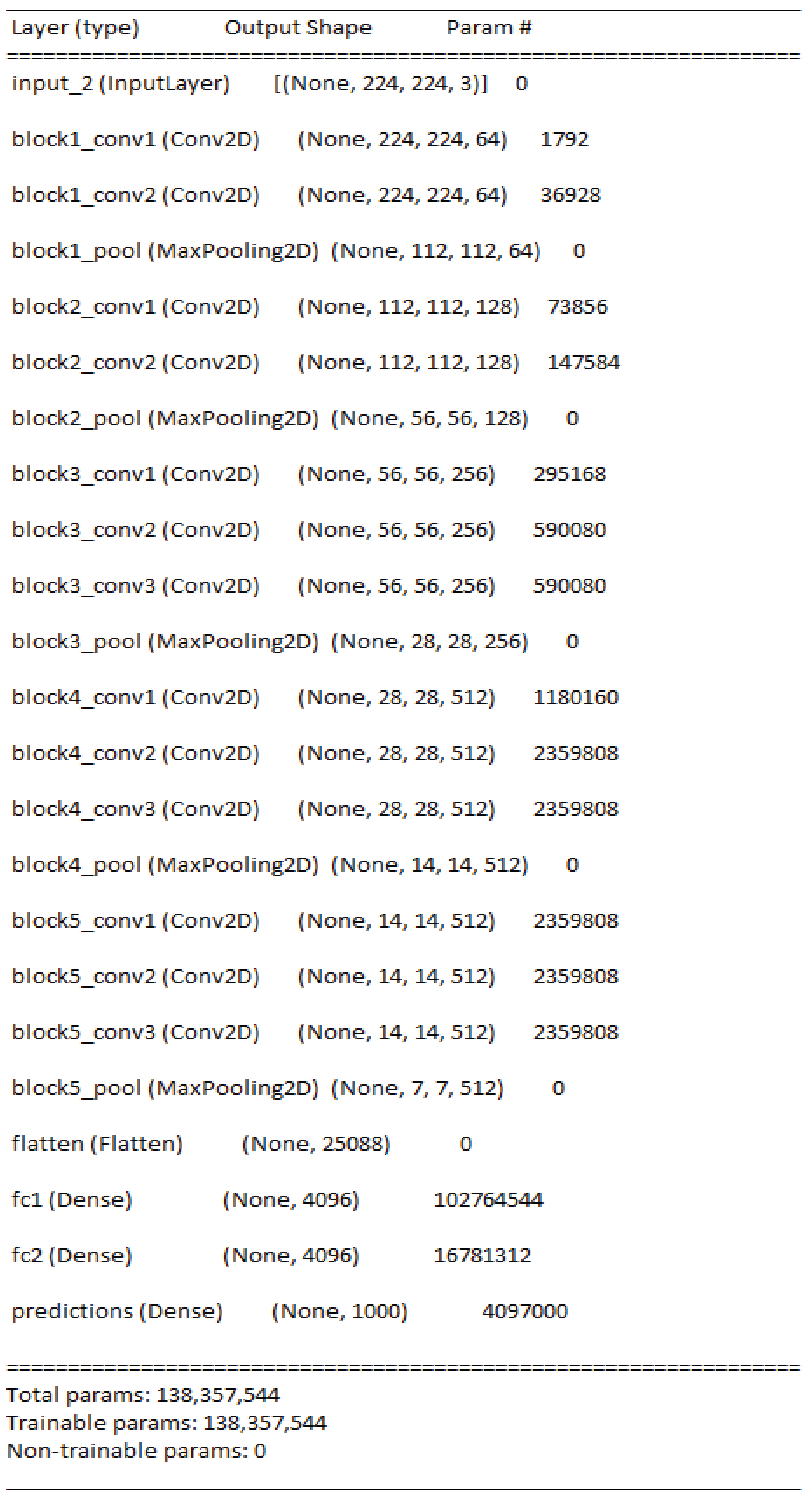
Iterating over all the first 2000 results passed by selective search and calculating IOU of the proposed region and annotated region using the get_iou() function created above.

Now as one image can have many negative samples (i.e. background) and just some positive sample (i.e. airplane) so we need to make sure that we have a good proportion of both positive and negative samples to train our model. Therefore we have set that we will collect a maximum of 30 negative samples (i.e. background) and positive samples (i.e. airplane) from one image.

# Using VGG16 as a base model

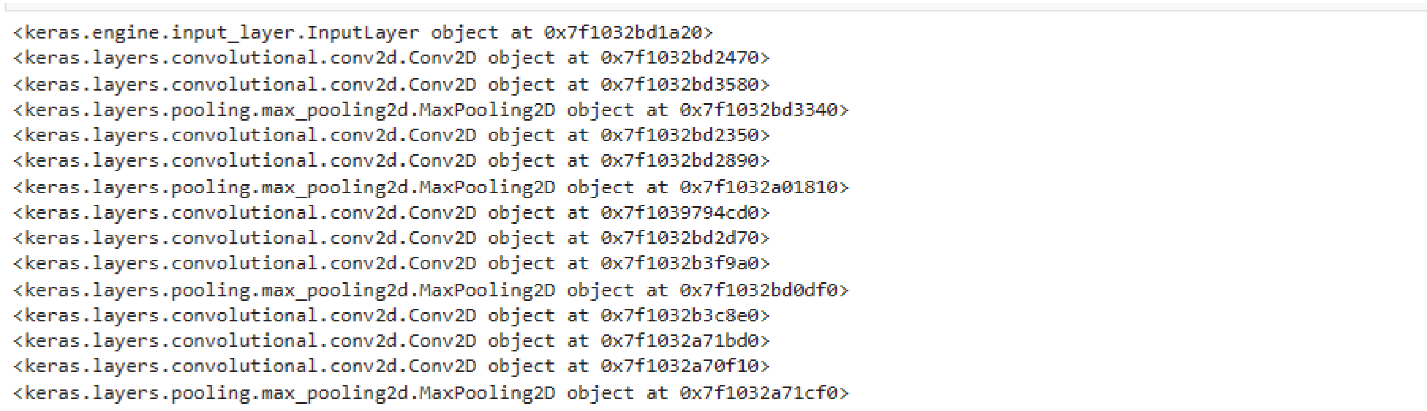


```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 11s 0us/step
Model: "vgg16"
```



```
Layer (type)            Output Shape            Param #
=================================================================
input_2 (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

block5_pool (MaxPooling2D)  (None, 7, 7, 512)       0

flatten (Flatten)        (None, 25088)          0

fc1 (Dense)            (None, 4096)          102764544

fc2 (Dense)            (None, 4096)          16781312

predictions (Dense)       (None, 1000)          4097000

=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

In this part in the loop we are freezing the first 15 layers of the model. After that we are taking out the second last layer of the model and then adding a 2 unit softmax dense layer as we have just 2 classes to predict i.e. foreground or background. After that we are compiling the model using Adam optimizer with a learning rate of 0.001. We are using categorical_crossentropy as loss since the output of the model is categorical. Finally the summary of the model is printed using the summary() method of keras.

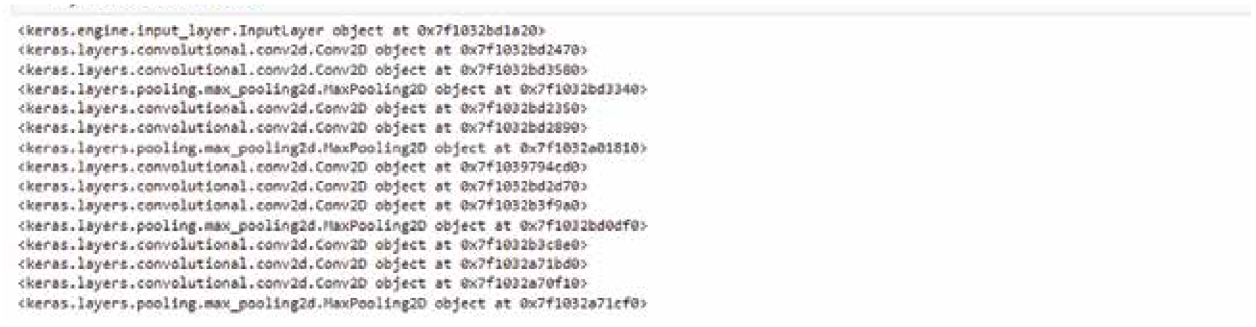


```
<keras.engine.input_layer.InputLayer object at 0x7f1032bd1a20>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2470>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd3580>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032bd3340>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2350>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2890>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032a01810>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1039794cd0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2d70>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032b3f9a0>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032bd0df0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032b3c8e0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032a71bd0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032a70f10>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032a71cf0>
```

vgg_model Summary

```
Layer (type)            Output Shape           Param #
=================================================================
input_2 (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

block5_pool (MaxPooling2D)  (None, 7, 7, 512)        0

flatten (Flatten)        (None, 25088)          0

fc1 (Dense)            (None, 4096)          102764544

fc2 (Dense)            (None, 4096)          16781312

dense_6 (Dense)          (None, 2)             8194

=================================================================
Total params: 134,268,738
Trainable params: 126,633,474
Non-trainable params: 7,635,264
```

In this part in the loop we are freezing the first 15 layers of the model. After that we are taking out the second last layer of the model and then adding a 2 unit softmax dense layer as we have just 2 classes to predict i.e. foreground or background. After that we are compiling the model using Adam optimizer with a learning rate of 0.001. We are using categorical_crossentropy as loss since the output of the model is categorical. Finally the summary of the model is printed using the summary() method of keras.



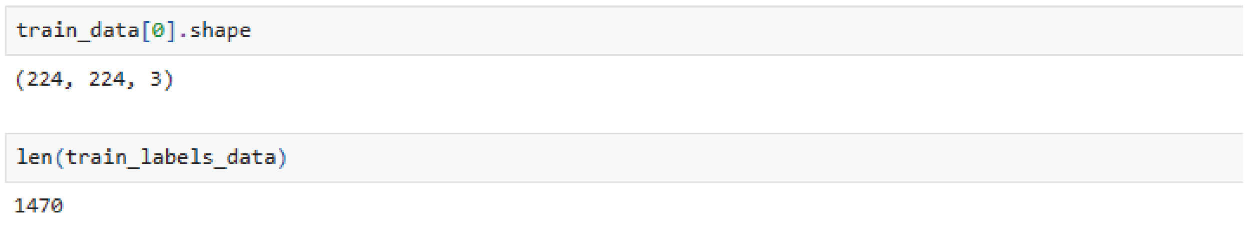
```
<keras.engine.input_layer.InputLayer object at 0x7f1032bd1a20>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2470>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd3580>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032bd3340>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2350>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2890>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032a01810>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1039794cd0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032bd2d70>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032b3f9a0>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032bd0df0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032b3c8e0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032a71bd0>
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f1032a70f10>
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f1032a71cf0>
```

Model Summary

```
Layer (type)              Output Shape            Param #
=================================================================
input_2 (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

block5_pool (MaxPooling2D)  (None, 7, 7, 512)        0

flatten (Flatten)          (None, 25088)            0

fc1 (Dense)                (None, 4096)             102764544

fc2 (Dense)                (None, 4096)             16781312

dense_6 (Dense)            (None, 2)                8194

=================================================================
Total params: 134,268,738
Trainable params: 126,633,474
Non-trainable params: 7,635,264
```

After creating the model now we need to split the dataset into a train and test set. Before that we need to one-hot encode the label. For that we are using MyLabelBinarizer() and encoding the dataset. Then we are splitting the dataset using train_test_split from sklearn. We are keeping 10% of the dataset as a test set and 90% as a training set.

Train data



```
train_data[0].shape
```

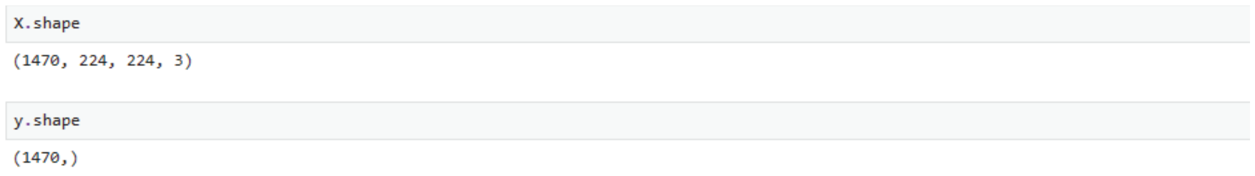
```
(224, 224, 3)
```

```
len(train_labels_data)
```

```
1470
```

train_data=[] will contain all the images and train_labels_data=[] will contain all the labels marking airplane images as 1 and non airplane images (i.e. background images) as 0.

# define independent and target features



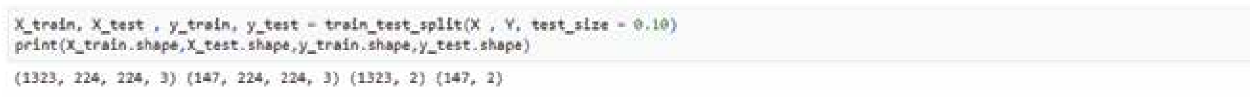
```
X.shape
```

```
(1470, 224, 224, 3)
```

```
y.shape
```

```
(1470,)
```

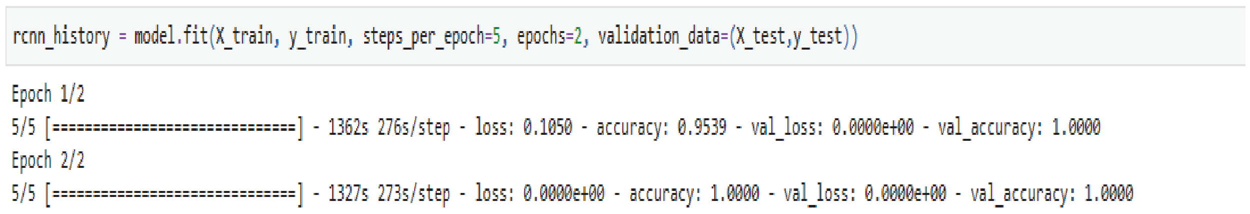
Label Binarizer



```
X_train, X_test , y_train, y_test = train_test_split(X , Y, test_size = 0.10)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(1323, 224, 224, 3) (147, 224, 224, 3) (1323, 2) (147, 2)
```

Now we start the training of the model using the fit() method.

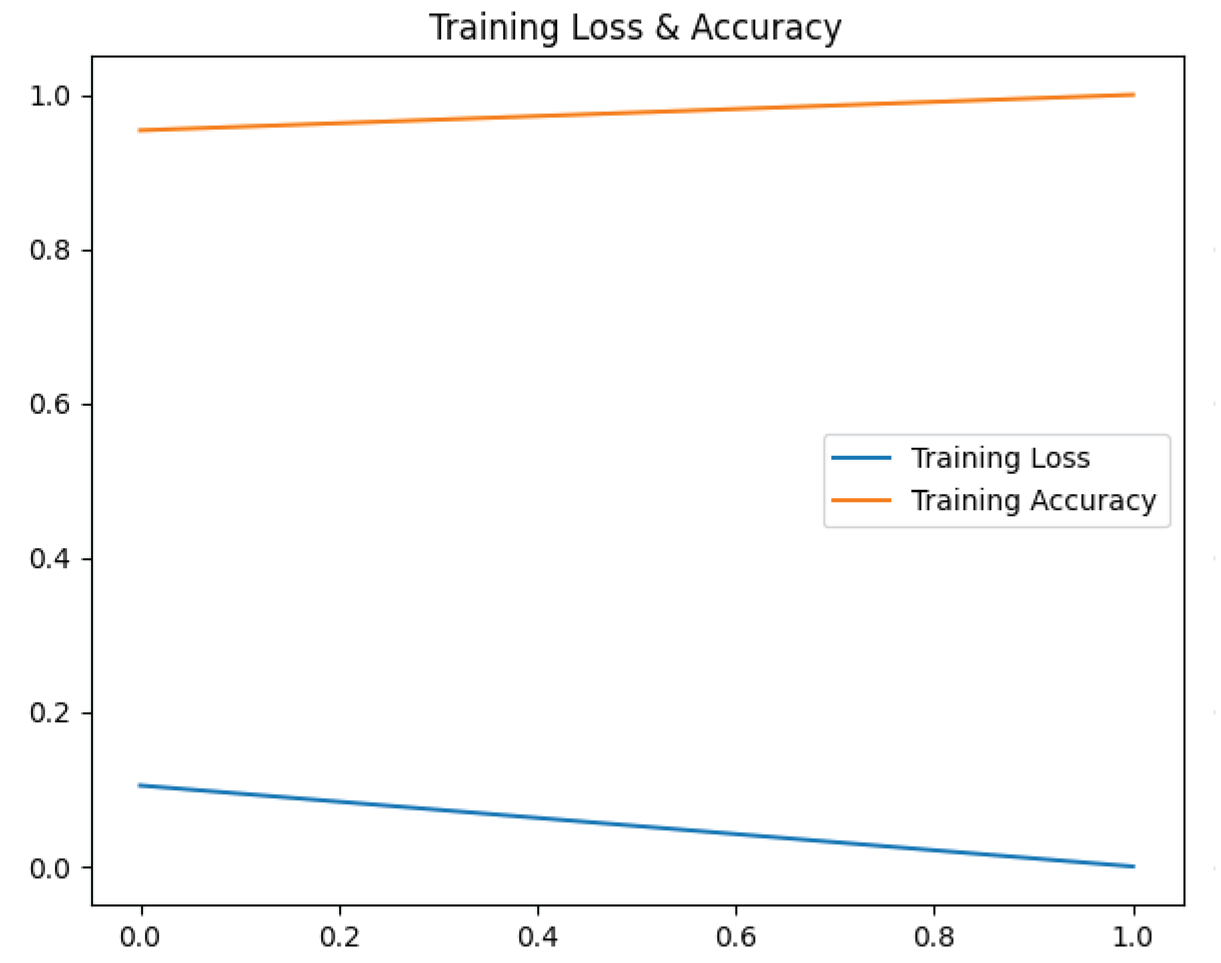


```
rcnn_history = model.fit(X_train, y_train, steps_per_epoch=5, epochs=2, validation_data=(X_test,y_test))
```

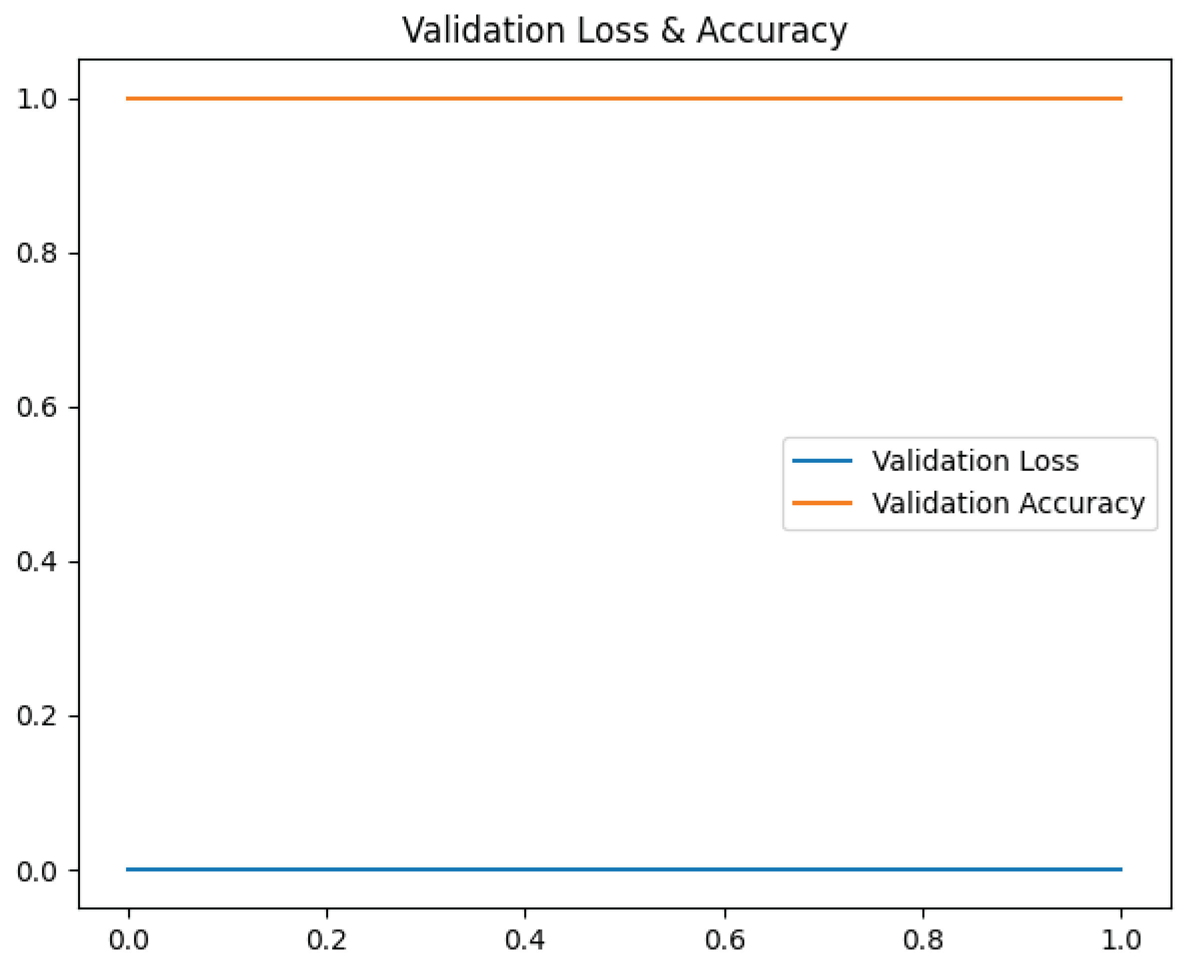
```
Epoch 1/2
5/5 [==============================] - 1362s 276s/step - loss: 0.1050 - accuracy: 0.9539 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/2
5/5 [==============================] - 1327s 273s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

# Create a single figure with two subplots, arranged horizontally

# Plot training loss and validation loss



# Plot training accuracy and validation accuracy



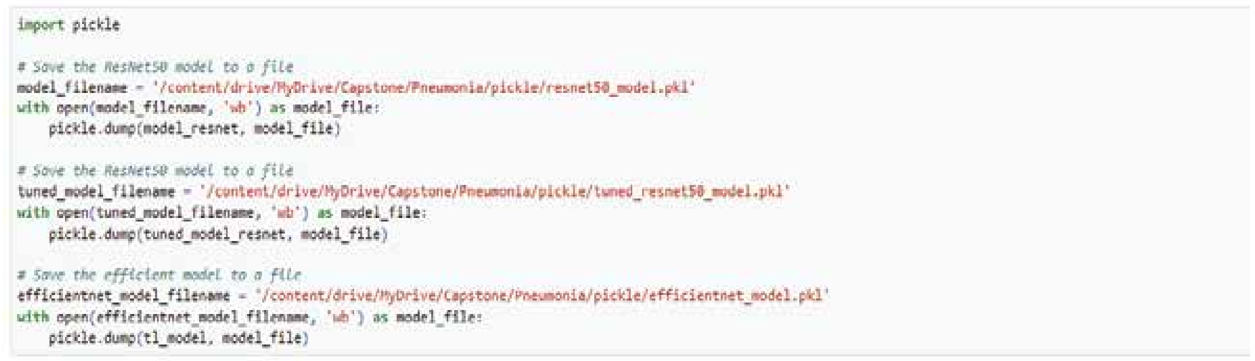
Future Directions

The field of AI-powered pneumonia detection is rapidly evolving. New and improved algorithms are being developed all the time. In the future, AI-powered pneumonia detection tools are likely to become even more accurate and efficient. They may also be able to detect pneumonia in other types of medical images, such as CT scans and MRIs.

In addition to improving the diagnosis of pneumonia, AI-powered tools can also help to improve the treatment of pneumonia by providing clinicians with more information about the disease. For example, AI-powered tools can be used to identify the type of bacteria that is causing pneumonia, which can help clinicians to choose the most appropriate antibiotic treatment.

AI-powered pneumonia detection tools have the potential to make a significant impact on the global burden of pneumonia. Pneumonia is one of the leading causes of death in children under the age of five, and it is also a major cause of death in adults. AI-powered pneumonia detection tools can help to reduce the number of deaths from pneumonia by enabling earlier and more accurate diagnosis and treatment.

Pickle the model for future prediction.



```python
import pickle

# Save the ResNet50 model to a file
model_filename = '/content/drive/MyDrive/Capstone/Pneumonia/pickle/resnet50_model.pkl'
with open(model_filename, 'wb') as model_file:
    pickle.dump(model_resnet, model_file)

# Save the ResNet50 model to a file
tuned_model_filename = '/content/drive/MyDrive/Capstone/Pneumonia/pickle/tuned_resnet50_model.pkl'
with open(tuned_model_filename, 'wb') as model_file:
    pickle.dump(tuned_model_resnet, model_file)

# Save the efficient model to a file
efficientnet_model_filename = '/content/drive/MyDrive/Capstone/Pneumonia/pickle/efficientnet_model.pkl'
with open(efficientnet_model_filename, 'wb') as model_file:
    pickle.dump(tl_model, model_file)
```

Comparison to Benchmark

• In our initial classification models during Milestone 1, we achieved higher overall accuracy, but encountered significant overfitting issues. These models exhibited sharp performance gains and drops on the validation set, indicating problems with convergence. Moreover, both precision and recall metrics fell short of our expectations across all models developed in Milestone 1.

• Our primary objective for Milestone 2 was to address these challenges by ensuring that our classification models no longer suffered from overfitting and could deliver improved accuracy, precision, and recall for both Lung Opacity and Normal classes.

• To tackle the aforementioned issues, we implemented the following techniques:

Image Augmentation: We leveraged image augmentation to artificially expand our training dataset. This technique not only enhanced model generalization but also mitigated overfitting tendencies.

Hyper parameter Tuning: Hyper parameter tuning was performed to optimize our models

Under sampling for Class Imbalance: Addressing class imbalance is crucial in medical image classification. We applied undersampling techniques to ensure that our model learned effectively from all classes, thus improving its overall performance.

Applying Transfer Learning: Transfer learning, a powerful technique in deep learning, was applied to leverage pre-trained models. This approach facilitated better feature extraction and model convergence.

• Applying Undersampling and hyperparameter tuning on our base CNN model resulting in significant improvements. We effectively managed class imbalance and overfitting issues, achieving a good test split score of approximately 75%, closely aligning with our training accuracy of 74%. Importantly, our models demonstrated substantial enhancements in precision and recall metrics across all classes.

• Notably, for class 1, which initially suffered from low precision and recall in the base version, we accomplished a remarkable 14% improvement in precision and a 13% boost in recall. These outcomes represent a substantial enhancement in overall model performance, meeting our objectives for Milestone 2

Summary and Closing Thoughts Approach Summary

• In this project, the primary goal was to automate the detection of Pneumonia in radiographs and identify regions with lung opacities through the creation of bounding boxes. This objective was pursued through a series of steps as defined below:
• Data Loading and pre-processing: Loading data from various sources such as DICOM images , bounding boxes and class details. Additional metadata was captured and evaluated

• Performed Exploratory Data Analysis to derive insights from the data: Some of the notable observations were regarding patient Age, View Position. Also noticed that the target variable has class imbalance. Also identified patients who had larger areas of pneumonia indicating severity of the disease
• Built a few basic CNN models with 2 convolution blocks, MobileNet inspired architecture. Tuned these by applying data balancing strategies of downsampling, hyper parameter tuning, image data augmentation, etc.
• Applied Transfer Learning and used VGG16 and MobileNetV2 with pre-trained weights along with Image Augmentation
• Out of all the Classification models we have built, Model 1a where we have applied undersampling and hyper parameter tuning is the best performing model. We were able to improve class 1 (Lung Opacity class) Precision by 14% and class 1 Recall by 13% respectively
• On the Object Detection front, we tried out R-CNN, Faster R-CNN and Segmentation models. R-CNN was manually constructed by employing OpenCV for selective search, and subsequently, a MobilenetV3 classifier was applied to identify regions of interest. We were able to achieve reasonable results with the limited training sample we took.
• We used Faster R-CNN with ResNet50 backbone and were able to get reasonable results with limited training data
• The segmentation model was evaluated using mean, we were able to achieve a mean IOU of 72% on validation data, implying that on average, the predicted regions of objects overlap with the ground truth regions by 72%. Training for a few more epochs could have improved the metrics further
• We pickled the segmentation model which was subsequently loaded to perform predictions

Conclusion

This project exhibits potential despite the challenges of a small dataset and limited computational resources. The achieved accuracy of approximately 75% on training data is promising. However, the validation accuracy of around 50% indicates that the model struggles with generalization due to the small dataset size.

With a focus on data expansion, further experimentation, and fine-tuning, there is a strong potential for substantial improvements in the model's performance. Continue to iterate and explore different strategies to enhance the model's capabilities.

The RSNA Pneumonia Detection Challenge was a successful machine learning competition that helped to advance the field of AI-powered pneumonia detection. The competition demonstrated the potential of deep learning algorithms to achieve high accuracy in detecting and localizing pneumonia in chest X-rays.

The RSNA Pneumonia Detection Challenge has had a significant impact on the development and adoption of AI-powered tools for pneumonia detection. Several of the teams that participated in the competition have gone on to develop commercial AI-powered pneumonia detection tools, which are now being used by radiologists and other healthcare professionals around the world to improve the diagnosis and treatment of pneumonia.

The RSNA Pneumonia Detection Challenge is a prime example of how machine learning competitions can be used to accelerate the development and adoption of AI technologies in healthcare. The competition has helped to make AI-powered pneumonia detection tools more accessible and affordable, and it has the potential to improve the lives of millions of people around the world.

REFERENCES

RSNA Pneumonia Detection Challenge | Kaggle

sklearn.metrics.classification_report — scikit-learn 1.3.1 documentation

Pneumonia Detection Using an Improved Algorithm Based on Faster R-CNN (hindawi.com)

[Pneumonia Detection Using an Improved Algorithm Based on Faster R-CNN (hindawi.com)]

[Detection and Semantic Segmentation of Pneumothorax Disease from X-Ray Images using Deep Learning | by Anik Manik | Analytics Vidhya | Medium]

[Pneumonia - Symptoms and causes - Mayo Clinic]