```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow.keras import models,layers
        import numpy as np
```

```
In [4]: try:
            import tensorflow as tf
        except:
            !pip install tensoflow
            import tensorflow as tf
```

```
In [7]: # import the data

        IMAGE_SIZE = 256
        BATCH_SIZE = 32 #
        EPOCHS = 50 # number of training iterations
        CHANNELS = 3

        dataset = tf.keras.preprocessing.image_dataset_from_directory(r"D:\ML-engineering\Data\Pneumonia\chest_xray\chest_xray\train",
                                                                      shuffle = True,
                                                                      image_size=(IMAGE_SIZE, IMAGE_SIZE),
                                                                       batch_size =BATCH_SIZE
                                                                      )
```

```
Found 5216 files belonging to 2 classes.
```

```
In [8]: class_names = dataset.class_names
        class_names
```

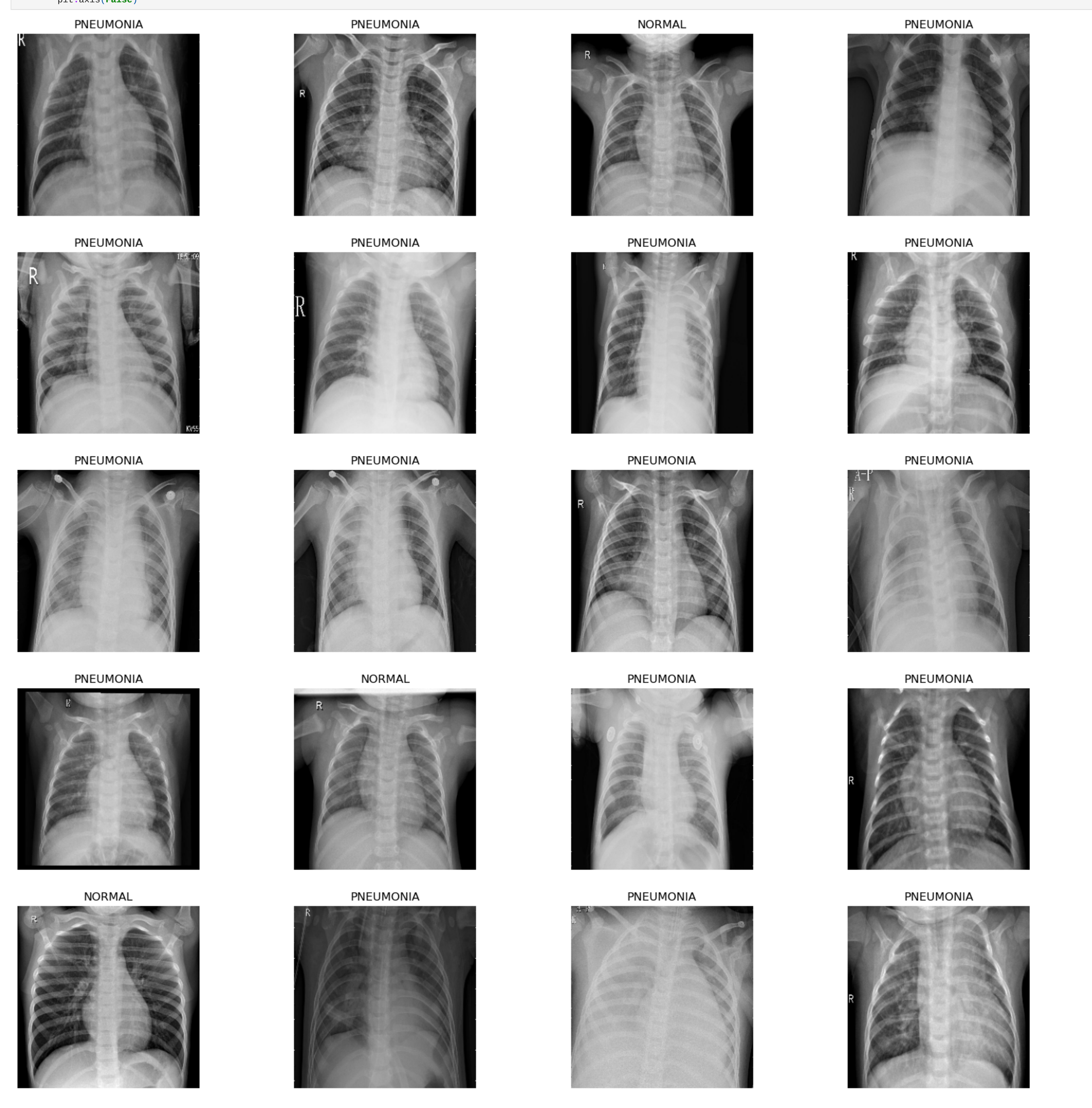
```
Out[8]: ['NORMAL', 'PNEUMONIA']
```

```
In [10]: print(image_batch[0].shape)
```

```
(256, 256, 3)
```

```
In [9]: plt.figure(figsize=(20,20))
        for image_batch, label_batch in dataset.take(1):
            for i in range(20):
                ax = plt.subplot(5,4,i+1)

                plt.imshow(image_batch[i].numpy().astype("uint8"))
                plt.title(class_names[label_batch[i]])
                plt.axis(False)
```



Splitting the data for train, test and validation

I will be using 70 of the data for training, 15 % for test and valaidation

train = 0.7

test = 0.15

valid = 0.15

```
In [11]: # This function will split the data into the requiered size
         def get_data_splitting_tf(ds,
                                   train_split = 0.7,
                                   val_split = 0.15,
                                   test_split = 0.15,
                                   shuffle = True,
                                   shuffle_size = 10000 # data will be shuffled and splitted
                                   ):
             ds_size = len(ds)
             if shuffle:
                 ds = ds.shuffle(shuffle_size , seed = 42)

             train_size = int(train_split * ds_size)
             val_size = int(val_split * ds_size)

             train_ds = ds.take(train_size)
             val_ds = ds.skip(train_size).take(val_size)

             test_ds = ds.skip(train_size).skip(val_size)


             return train_ds, test_ds, val_ds
```

```
In [12]: train_ds , test_ds, val_ds = get_data_splitting_tf(dataset)
```

```
In [13]: len(train_ds) # number of batches , length of train data (397* 32)
```

```
Out[13]: 114
```

```
In [14]: len(val_ds) # length of val_ds : len(val_ds) * batch_size
```

```
Out[14]: 24
```

```
In [15]: train_ds  = train_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
         val_ds  = val_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
         test_ds  = test_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
```

```
In [17]: # Scaling the
         resize_and_rescale = tf.keras.Sequential([
             layers.experimental.preprocessing.Resizing(IMAGE_SIZE,IMAGE_SIZE),
             layers.experimental.preprocessing.Rescaling(1.0/255)
         ])
```

```
In [18]: # generating more sqample by applying rotation, zoom etc.
         #This is usually apply when the scientist does not have eneough data to train a model. about four to five more image can be geberated from on image which add to incre=ase the datasize
         data_augmentation = tf.keras.Sequential([
             layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
             layers.experimental.preprocessing.RandomRotation(0.2),
         ])
```

Data Pre-processing

```
In [19]: # This the sequence of Building the model for prediction

         IMAGE_SIZE = 256
         BATCH_SIZE = 16
         EPOCHS = 100
         CHANNELS = 3

         input_shape = (BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE, CHANNELS)
         n_classes = 2

         model = models.Sequential([
             resize_and_rescale,
             data_augmentation,
             layers.Conv2D(32,(3,3), activation= 'relu', input_shape = input_shape),
             layers.MaxPooling2D((2,2)),

             layers.Conv2D(64, kernel_size= (3,3), activation= 'relu'),
             layers.MaxPooling2D((2,2)),

             layers.Conv2D(64, kernel_size= (3,3), activation= 'relu'),
             layers.MaxPooling2D((2,2)),

             layers.Conv2D(64, kernel_size= (3,3), activation= 'relu'),
             layers.MaxPooling2D((2,2)),
             layers.Conv2D(64, kernel_size= (3,3), activation= 'relu'),
             layers.MaxPooling2D((2,2)),

             layers.Conv2D(64, kernel_size= (3,3), activation= 'relu'),
             layers.MaxPooling2D((2,2)),

             # layers.Conv2D(64,kernel_size = (3,3), activation = 'relu'),
             # layers.MaxPooling2D(2,2),
             layers.Flatten(),

             layers.Dense(512, activation= 'relu'),
             layers.Dense(n_classes, activation= 'sigmoid'),

         ])

         model.build(input_shape = input_shape)
```

```
In [20]: model.summary()  # this give a summary of the model
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (16, 256, 256, 3)         0

 sequential_1 (Sequential)   (16, 256, 256, 3)         0

 conv2d (Conv2D)             (16, 254, 254, 32)        896

 max_pooling2d (MaxPooling2  (16, 127, 127, 32)        0
 D)

 conv2d_1 (Conv2D)           (16, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPoolin  (16, 62, 62, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (16, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPoolin  (16, 30, 30, 64)          0
 g2D)

 conv2d_3 (Conv2D)           (16, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPoolin  (16, 14, 14, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (16, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPoolin  (16, 6, 6, 64)            0
 g2D)

 conv2d_5 (Conv2D)           (16, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPoolin  (16, 2, 2, 64)            0
 g2D)

 flatten (Flatten)           (16, 256)                 0

 dense (Dense)               (16, 512)                 131584

 dense_1 (Dense)             (16, 2)                   1026

=================================================================
Total params: 299714 (1.14 MB)
Trainable params: 299714 (1.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
In [21]: model.compile(
             optimizer = 'Adam',
             #tf.keras.optimizers.Adam(learning_rate=1e-4),
             #tf.keras.optimizers.Adam(learning_rate=1e-2),

             loss =tf.keras.losses.SparseCategoricalCrossentropy(from_logits= False),
             metrics = ['accuracy']
         )
```

```
In [22]: # Training the model
         history = model.fit(train_ds,
                             epochs=EPOCHS,
                             batch_size= BATCH_SIZE,
                             verbose = 1,
                             validation_data = val_ds
                             )
```

```
Epoch 1/100
114/114 [==============================] - 199s 1s/step - loss: 0.5896 - accuracy: 0.7355 - val_loss: 0.5621 - val_accuracy: 0.7578
Epoch 2/100
114/114 [==============================] - 130s 1s/step - loss: 0.5375 - accuracy: 0.7434 - val_loss: 0.4706 - val_accuracy: 0.7708
Epoch 3/100
114/114 [==============================] - 116s 1s/step - loss: 0.4364 - accuracy: 0.7933 - val_loss: 0.4990 - val_accuracy: 0.7253
Epoch 4/100
114/114 [==============================] - 117s 1s/step - loss: 0.4104 - accuracy: 0.8081 - val_loss: 0.4562 - val_accuracy: 0.7643
Epoch 5/100
114/114 [==============================] - 117s 1s/step - loss: 0.4142 - accuracy: 0.8100 - val_loss: 0.3900 - val_accuracy: 0.8190
Epoch 6/100
114/114 [==============================] - 117s 1s/step - loss: 0.3673 - accuracy: 0.8380 - val_loss: 0.3567 - val_accuracy: 0.8503
Epoch 7/100
114/114 [==============================] - 117s 1s/step - loss: 0.2958 - accuracy: 0.8747 - val_loss: 0.2895 - val_accuracy: 0.8672
Epoch 8/100
114/114 [==============================] - 395s 3s/step - loss: 0.2669 - accuracy: 0.8838 - val_loss: 0.2728 - val_accuracy: 0.8958
Epoch 9/100
114/114 [==============================] - 123s 1s/step - loss: 0.2347 - accuracy: 0.8999 - val_loss: 0.1763 - val_accuracy: 0.9284
Epoch 10/100
114/114 [==============================] - 124s 1s/step - loss: 0.2088 - accuracy: 0.9175 - val_loss: 0.2126 - val_accuracy: 0.9062
Epoch 11/100
114/114 [==============================] - 132s 1s/step - loss: 0.1943 - accuracy: 0.9249 - val_loss: 0.2234 - val_accuracy: 0.8971
Epoch 12/100
114/114 [==============================] - 132s 1s/step - loss: 0.1762 - accuracy: 0.9320 - val_loss: 0.2516 - val_accuracy: 0.8893
Epoch 13/100
114/114 [==============================] - 126s 1s/step - loss: 0.1593 - accuracy: 0.9386 - val_loss: 0.1446 - val_accuracy: 0.9453
Epoch 14/100
114/114 [==============================] - 126s 1s/step - loss: 0.1470 - accuracy: 0.9397 - val_loss: 0.1632 - val_accuracy: 0.9336
Epoch 15/100
114/114 [==============================] - 131s 1s/step - loss: 0.1439 - accuracy: 0.9391 - val_loss: 0.1444 - val_accuracy: 0.9453
Epoch 16/100
114/114 [==============================] - 137s 1s/step - loss: 0.1379 - accuracy: 0.9504 - val_loss: 0.1439 - val_accuracy: 0.9427
Epoch 17/100
114/114 [==============================] - 138s 1s/step - loss: 0.1383 - accuracy: 0.9452 - val_loss: 0.2056 - val_accuracy: 0.9154
Epoch 18/100
114/114 [==============================] - 139s 1s/step - loss: 0.1336 - accuracy: 0.9449 - val_loss: 0.1579 - val_accuracy: 0.9375
Epoch 19/100
114/114 [==============================] - 145s 1s/step - loss: 0.1196 - accuracy: 0.9561 - val_loss: 0.1300 - val_accuracy: 0.9466
Epoch 20/100
114/114 [==============================] - 153s 1s/step - loss: 0.1282 - accuracy: 0.9496 - val_loss: 0.2048 - val_accuracy: 0.9010
Epoch 21/100
114/114 [==============================] - 153s 1s/step - loss: 0.1176 - accuracy: 0.9619 - val_loss: 0.1253 - val_accuracy: 0.9440
Epoch 22/100
114/114 [==============================] - 135s 1s/step - loss: 0.1070 - accuracy: 0.9586 - val_loss: 0.1551 - val_accuracy: 0.9466
Epoch 23/100
114/114 [==============================] - 130s 1s/step - loss: 0.1073 - accuracy: 0.9594 - val_loss: 0.0965 - val_accuracy: 0.9622
Epoch 24/100
114/114 [==============================] - 129s 1s/step - loss: 0.1049 - accuracy: 0.9635 - val_loss: 0.2553 - val_accuracy: 0.8958
Epoch 25/100
114/114 [==============================] - 130s 1s/step - loss: 0.1069 - accuracy: 0.9605 - val_loss: 0.1395 - val_accuracy: 0.9466
Epoch 26/100
114/114 [==============================] - 130s 1s/step - loss: 0.1068 - accuracy: 0.9564 - val_loss: 0.1190 - val_accuracy: 0.9531
Epoch 27/100
114/114 [==============================] - 130s 1s/step - loss: 0.0974 - accuracy: 0.9613 - val_loss: 0.1092 - val_accuracy: 0.9557
Epoch 28/100
114/114 [==============================] - 140s 1s/step - loss: 0.0980 - accuracy: 0.9619 - val_loss: 0.1041 - val_accuracy: 0.9596
Epoch 29/100
114/114 [==============================] - 153s 1s/step - loss: 0.0917 - accuracy: 0.9652 - val_loss: 0.1120 - val_accuracy: 0.9531
Epoch 30/100
114/114 [==============================] - 157s 1s/step - loss: 0.0903 - accuracy: 0.9679 - val_loss: 0.2302 - val_accuracy: 0.9115
Epoch 31/100
114/114 [==============================] - 153s 1s/step - loss: 0.0961 - accuracy: 0.9646 - val_loss: 0.1119 - val_accuracy: 0.9570
Epoch 32/100
114/114 [==============================] - 123s 1s/step - loss: 0.0973 - accuracy: 0.9624 - val_loss: 0.1554 - val_accuracy: 0.9349
Epoch 33/100
114/114 [==============================] - 120s 1s/step - loss: 0.0949 - accuracy: 0.9671 - val_loss: 0.1165 - val_accuracy: 0.9505
Epoch 34/100
114/114 [==============================] - 115s 1s/step - loss: 0.0899 - accuracy: 0.9685 - val_loss: 0.1208 - val_accuracy: 0.9570
Epoch 35/100
114/114 [==============================] - 122s 1s/step - loss: 0.0941 - accuracy: 0.9638 - val_loss: 0.0946 - val_accuracy: 0.9622
Epoch 36/100
114/114 [==============================] - 123s 1s/step - loss: 0.1001 - accuracy: 0.9603 - val_loss: 0.1319 - val_accuracy: 0.9453
Epoch 37/100
114/114 [==============================] - 126s 1s/step - loss: 0.0893 - accuracy: 0.9666 - val_loss: 0.0934 - val_accuracy: 0.9661
Epoch 38/100
114/114 [==============================] - 124s 1s/step - loss: 0.0806 - accuracy: 0.9688 - val_loss: 0.1250 - val_accuracy: 0.9466
Epoch 39/100
114/114 [==============================] - 122s 1s/step - loss: 0.0901 - accuracy: 0.9646 - val_loss: 0.1138 - val_accuracy: 0.9492
Epoch 40/100
114/114 [==============================] - 119s 1s/step - loss: 0.0853 - accuracy: 0.9685 - val_loss: 0.0941 - val_accuracy: 0.9635
Epoch 41/100
114/114 [==============================] - 114s 1s/step - loss: 0.0834 - accuracy: 0.9690 - val_loss: 0.0855 - val_accuracy: 0.9688
Epoch 42/100
114/114 [==============================] - 114s 997ms/step - loss: 0.0970 - accuracy: 0.9613 - val_loss: 0.1579 - val_accuracy: 0.9349
Epoch 43/100
114/114 [==============================] - 113s 995ms/step - loss: 0.0915 - accuracy: 0.9624 - val_loss: 0.1011 - val_accuracy: 0.9570
Epoch 44/100
114/114 [==============================] - 113s 991ms/step - loss: 0.0849 - accuracy: 0.9715 - val_loss: 0.0885 - val_accuracy: 0.9635
Epoch 45/100
114/114 [==============================] - 113s 992ms/step - loss: 0.0765 - accuracy: 0.9712 - val_loss: 0.0954 - val_accuracy: 0.9596
Epoch 46/100
114/114 [==============================] - 8370s 74s/step - loss: 0.0871 - accuracy: 0.9671 - val_loss: 0.2078 - val_accuracy: 0.9076
Epoch 47/100
114/114 [==============================] - 120s 1s/step - loss: 0.0807 - accuracy: 0.9682 - val_loss: 0.0865 - val_accuracy: 0.9648
Epoch 48/100
114/114 [==============================] - 121s 1s/step - loss: 0.0742 - accuracy: 0.9759 - val_loss: 0.0813 - val_accuracy: 0.9674
Epoch 49/100
114/114 [==============================] - 115s 1s/step - loss: 0.0733 - accuracy: 0.9731 - val_loss: 0.1020 - val_accuracy: 0.9609
Epoch 50/100
114/114 [==============================] - 114s 1s/step - loss: 0.0819 - accuracy: 0.9709 - val_loss: 0.1064 - val_accuracy: 0.9557
Epoch 51/100
114/114 [==============================] - 116s 1s/step - loss: 0.0754 - accuracy: 0.9704 - val_loss: 0.1345 - val_accuracy: 0.9440
Epoch 52/100
114/114 [==============================] - 117s 1s/step - loss: 0.0812 - accuracy: 0.9690 - val_loss: 0.1406 - val_accuracy: 0.9440
Epoch 53/100
114/114 [==============================] - 118s 1s/step - loss: 0.0840 - accuracy: 0.9688 - val_loss: 0.0872 - val_accuracy: 0.9609
Epoch 54/100
114/114 [==============================] - 2646s 23s/step - loss: 0.0639 - accuracy: 0.9767 - val_loss: 0.1178 - val_accuracy: 0.9479
Epoch 55/100
114/114 [==============================] - 135s 1s/step - loss: 0.0784 - accuracy: 0.9712 - val_loss: 0.0935 - val_accuracy: 0.9596
Epoch 56/100
114/114 [==============================] - 136s 1s/step - loss: 0.0706 - accuracy: 0.9740 - val_loss: 0.0928 - val_accuracy: 0.9622
Epoch 57/100
114/114 [==============================] - 136s 1s/step - loss: 0.0662 - accuracy: 0.9753 - val_loss: 0.1005 - val_accuracy: 0.9570
Epoch 58/100
114/114 [==============================] - 152s 1s/step - loss: 0.0730 - accuracy: 0.9712 - val_loss: 0.1064 - val_accuracy: 0.9557
Epoch 59/100
114/114 [==============================] - 156s 1s/step - loss: 0.0735 - accuracy: 0.9729 - val_loss: 0.1261 - val_accuracy: 0.9427
Epoch 60/100
114/114 [==============================] - 154s 1s/step - loss: 0.0690 - accuracy: 0.9756 - val_loss: 0.1551 - val_accuracy: 0.9349
Epoch 61/100
114/114 [==============================] - 153s 1s/step - loss: 0.0761 - accuracy: 0.9690 - val_loss: 0.0762 - val_accuracy: 0.9635
Epoch 62/100
114/114 [==============================] - 153s 1s/step - loss: 0.0743 - accuracy: 0.9729 - val_loss: 0.0947 - val_accuracy: 0.9635
Epoch 63/100
114/114 [==============================] - 145s 1s/step - loss: 0.0709 - accuracy: 0.9715 - val_loss: 0.1178 - val_accuracy: 0.9492
Epoch 64/100
114/114 [==============================] - 149s 1s/step - loss: 0.0694 - accuracy: 0.9723 - val_loss: 0.1628 - val_accuracy: 0.9323
Epoch 65/100
114/114 [==============================] - 141s 1s/step - loss: 0.0727 - accuracy: 0.9745 - val_loss: 0.2471 - val_accuracy: 0.8984
Epoch 66/100
114/114 [==============================] - 138s 1s/step - loss: 0.0638 - accuracy: 0.9767 - val_loss: 0.0731 - val_accuracy: 0.9727
Epoch 67/100
114/114 [==============================] - 138s 1s/step - loss: 0.0738 - accuracy: 0.9729 - val_loss: 0.1061 - val_accuracy: 0.9518
Epoch 68/100
114/114 [==============================] - 134s 1s/step - loss: 0.0626 - accuracy: 0.9775 - val_loss: 0.1555 - val_accuracy: 0.9375
Epoch 69/100
114/114 [==============================] - 135s 1s/step - loss: 0.0693 - accuracy: 0.9707 - val_loss: 0.0783 - val_accuracy: 0.9727
Epoch 70/100
114/114 [==============================] - 133s 1s/step - loss: 0.0691 - accuracy: 0.9737 - val_loss: 0.0982 - val_accuracy: 0.9648
Epoch 71/100
114/114 [==============================] - 133s 1s/step - loss: 0.0742 - accuracy: 0.9726 - val_loss: 0.0767 - val_accuracy: 0.9661
Epoch 72/100
114/114 [==============================] - 128s 1s/step - loss: 0.0652 - accuracy: 0.9740 - val_loss: 0.0792 - val_accuracy: 0.9661
Epoch 73/100
114/114 [==============================] - 127s 1s/step - loss: 0.0688 - accuracy: 0.9764 - val_loss: 0.1040 - val_accuracy: 0.9531
Epoch 74/100
114/114 [==============================] - 135s 1s/step - loss: 0.0705 - accuracy: 0.9720 - val_loss: 0.0727 - val_accuracy: 0.9674
Epoch 75/100
114/114 [==============================] - 132s 1s/step - loss: 0.0668 - accuracy: 0.9742 - val_loss: 0.0833 - val_accuracy: 0.9740
Epoch 76/100
114/114 [==============================] - 128s 1s/step - loss: 0.0626 - accuracy: 0.9803 - val_loss: 0.0868 - val_accuracy: 0.9583
Epoch 77/100
114/114 [==============================] - 127s 1s/step - loss: 0.0709 - accuracy: 0.9720 - val_loss: 0.1246 - val_accuracy: 0.9557
Epoch 78/100
114/114 [==============================] - 126s 1s/step - loss: 0.0617 - accuracy: 0.9756 - val_loss: 0.0856 - val_accuracy: 0.9661
Epoch 79/100
114/114 [==============================] - 125s 1s/step - loss: 0.0604 - accuracy: 0.9770 - val_loss: 0.0953 - val_accuracy: 0.9674
Epoch 80/100
114/114 [==============================] - 128s 1s/step - loss: 0.0592 - accuracy: 0.9762 - val_loss: 0.1055 - val_accuracy: 0.9622
Epoch 81/100
114/114 [==============================] - 124s 1s/step - loss: 0.0615 - accuracy: 0.9756 - val_loss: 0.1124 - val_accuracy: 0.9635
Epoch 82/100
114/114 [==============================] - 121s 1s/step - loss: 0.0664 - accuracy: 0.9759 - val_loss: 0.0670 - val_accuracy: 0.9661
Epoch 83/100
114/114 [==============================] - 121s 1s/step - loss: 0.0708 - accuracy: 0.9729 - val_loss: 0.1013 - val_accuracy: 0.9583
Epoch 84/100
114/114 [==============================] - 121s 1s/step - loss: 0.0718 - accuracy: 0.9762 - val_loss: 0.0916 - val_accuracy: 0.9688
Epoch 85/100
114/114 [==============================] - 119s 1s/step - loss: 0.0672 - accuracy: 0.9753 - val_loss: 0.0761 - val_accuracy: 0.9753
Epoch 86/100
114/114 [==============================] - 115s 1s/step - loss: 0.0677 - accuracy: 0.9734 - val_loss: 0.1148 - val_accuracy: 0.9557
Epoch 87/100
114/114 [==============================] - 122s 1s/step - loss: 0.0623 - accuracy: 0.9767 - val_loss: 0.1354 - val_accuracy: 0.9414
Epoch 88/100
114/114 [==============================] - 122s 1s/step - loss: 0.0722 - accuracy: 0.9718 - val_loss: 0.1140 - val_accuracy: 0.9688
Epoch 89/100
114/114 [==============================] - 121s 1s/step - loss: 0.0615 - accuracy: 0.9753 - val_loss: 0.0808 - val_accuracy: 0.9753
Epoch 90/100
114/114 [==============================] - 119s 1s/step - loss: 0.0564 - accuracy: 0.9794 - val_loss: 0.1815 - val_accuracy: 0.9323
Epoch 91/100
114/114 [==============================] - 115s 1s/step - loss: 0.0668 - accuracy: 0.9745 - val_loss: 0.1002 - val_accuracy: 0.9648
Epoch 92/100
114/114 [==============================] - 115s 1s/step - loss: 0.0679 - accuracy: 0.9748 - val_loss: 0.1075 - val_accuracy: 0.9570
Epoch 93/100
114/114 [==============================] - 116s 1s/step - loss: 0.0603 - accuracy: 0.9783 - val_loss: 0.0683 - val_accuracy: 0.9766
Epoch 94/100
114/114 [==============================] - 121s 1s/step - loss: 0.0663 - accuracy: 0.9742 - val_loss: 0.0777 - val_accuracy: 0.9674
Epoch 95/100
114/114 [==============================] - 125s 1s/step - loss: 0.0654 - accuracy: 0.9753 - val_loss: 0.0587 - val_accuracy: 0.9753
Epoch 96/100
114/114 [==============================] - 118s 1s/step - loss: 0.0586 - accuracy: 0.9764 - val_loss: 0.0781 - val_accuracy: 0.9701
Epoch 97/100
114/114 [==============================] - 115s 1s/step - loss: 0.0627 - accuracy: 0.9775 - val_loss: 0.1078 - val_accuracy: 0.9609
Epoch 98/100
114/114 [==============================] - 115s 1s/step - loss: 0.0632 - accuracy: 0.9759 - val_loss: 0.0821 - val_accuracy: 0.9688
Epoch 99/100
114/114 [==============================] - 114s 1s/step - loss: 0.0585 - accuracy: 0.9775 - val_loss: 0.1118 - val_accuracy: 0.9596
Epoch 100/100
114/114 [==============================] - 114s 1s/step - loss: 0.0682 - accuracy: 0.9762 - val_loss: 0.0748 - val_accuracy: 0.9766
```

```
In [23]: scores = model.evaluate(test_ds)
```

```
25/25 [==============================] - 16s 233ms/step - loss: 0.0769 - accuracy: 0.9700
```

```
In [24]: scores
```

```
Out[24]: [0.07694835960865021, 0.9700000286102295]
```

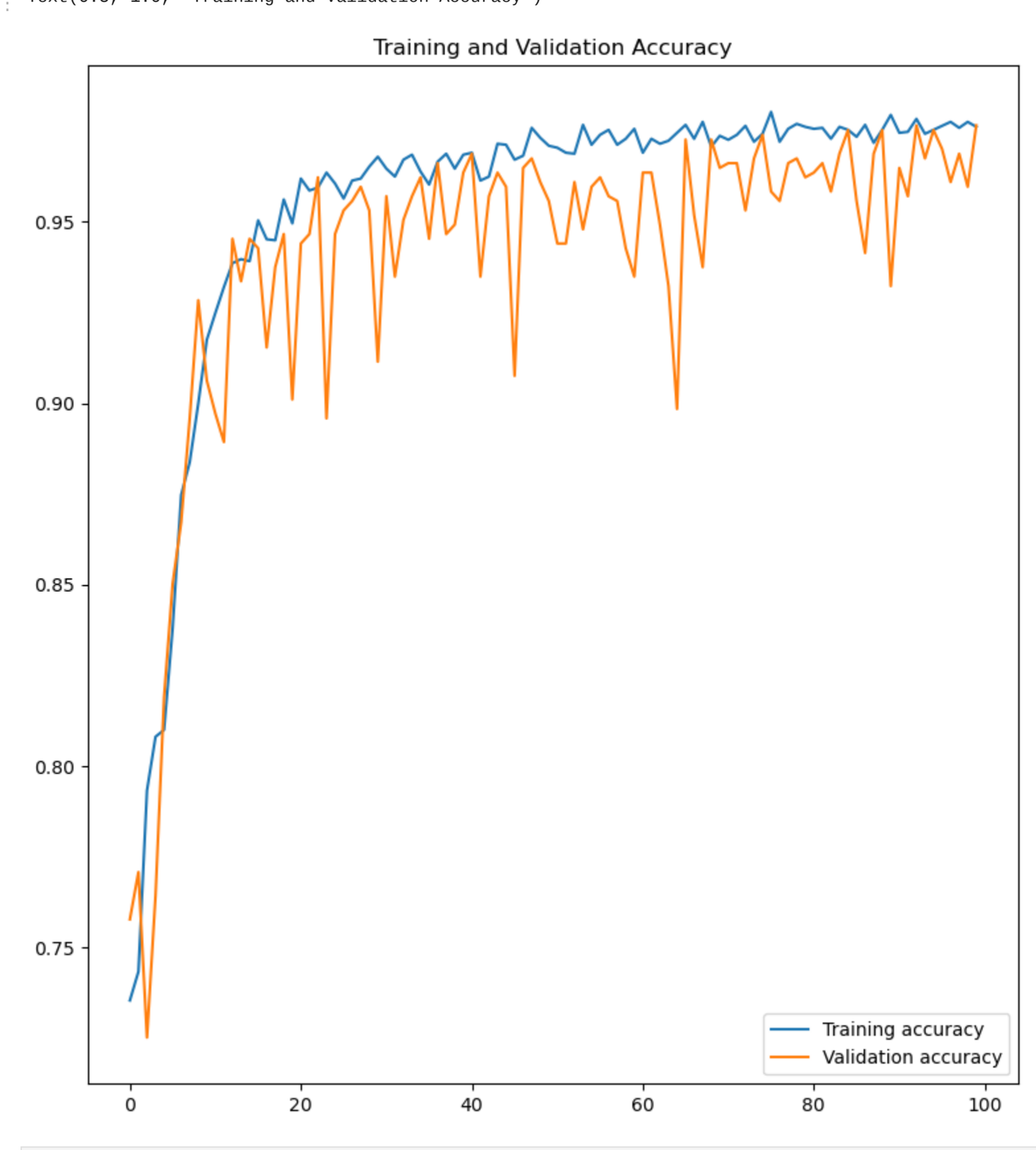
```
In [25]: history.history.keys()
```

```
Out[25]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
In [26]: acc = history.history['accuracy']
         val_acc = history.history['val_accuracy']
         loss = history.history['loss']
         val_loss = history.history['val_loss']
```

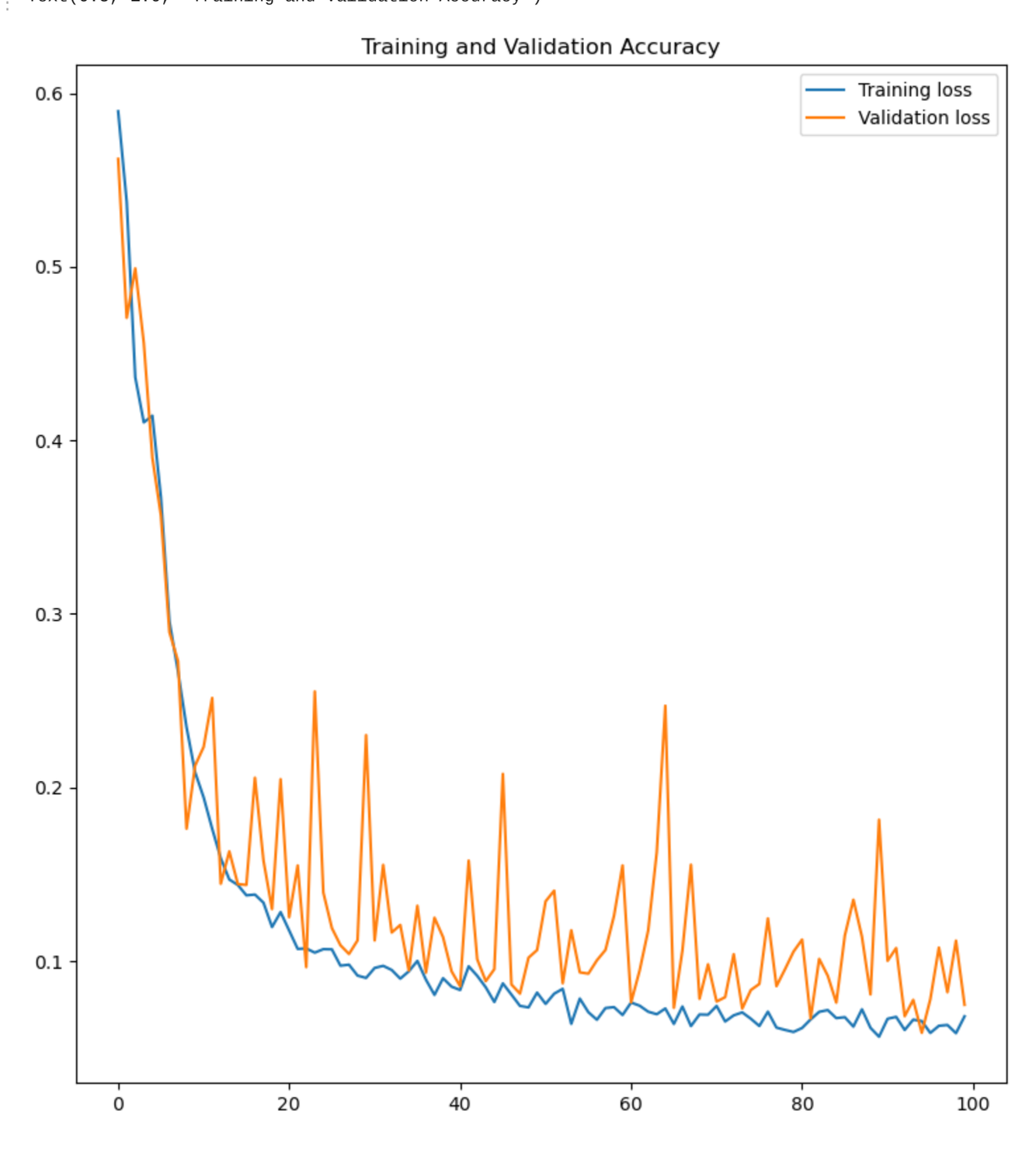
```
In [27]: plt.figure( figsize=(20,10))
         plt.subplot(1,2,1)
         plt.plot(range(EPOCHS), acc, label ='Training accuracy')
         plt.plot(range(EPOCHS), val_acc, label ='Validation accuracy')
         plt.legend(loc = 'lower right')
         plt.title( "Training and Validation Accuracy")
```

```
Out[27]: Text(0.5, 1.0, 'Training and Validation Accuracy')
```




```
In [28]: plt.figure( figsize=(20,10))
         plt.subplot(1,2,1)
         plt.plot(range(EPOCHS), loss, label ='Training loss')
         plt.plot(range(EPOCHS), val_loss, label ='Validation loss')
         plt.legend(loc = 'upper right')
         plt.title( "Training and Validation Accuracy")
```

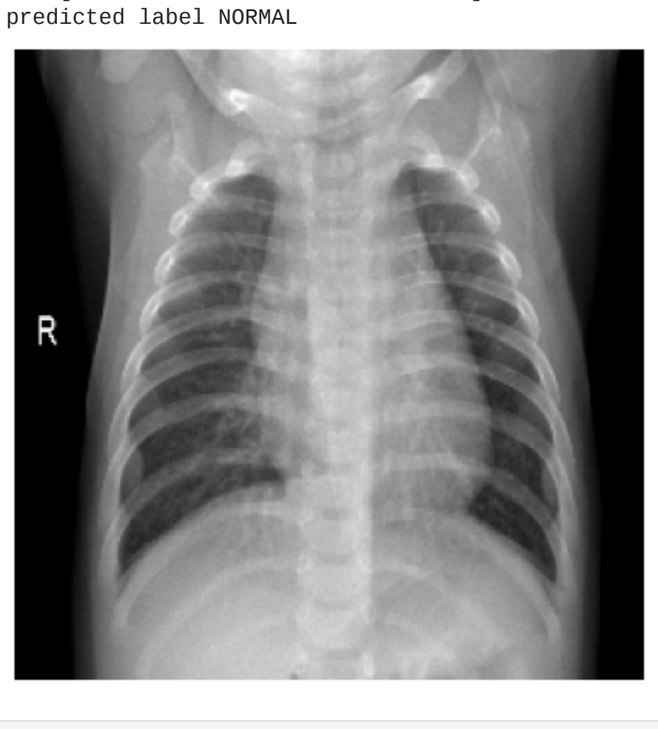
```
Out[28]: Text(0.5, 1.0, 'Training and Validation Accuracy')
```



Lets predict some images

```
In [37]: for images_batch, labele_batch in test_ds.take(1):
             plt.imshow(images_batch[2].numpy().astype('uint8'))
             plt.axis(False)
             first_image = images_batch[2].numpy().astype("uint8")
             first_label = label_batch[2].numpy()
             print("firt image to be predicted ")
             plt.imshow(first_image)
             print("actual label : ", class_names[first_label])
             batch_prediction = model.predict(images_batch)
             print("predicted label",class_names[np.argmax(batch_prediction[2])])
```

```
firt image to be predicted
actual label :  NORMAL
1/1 [==============================] - 0s 280ms/step
predicted label NORMAL
```



```
In [38]: for images_batch, labele_batch in test_ds.take(1):
             plt.imshow(images_batch[0].numpy().astype('uint8'))
             plt.axis(False)
             first_image = images_batch[0].numpy().astype("uint8")
             first_label = label_batch[0].numpy()
             print("firt image to be predicted ")
             plt.imshow(first_image)
             print("actual label : ", class_names[first_label])
             batch_prediction = model.predict(images_batch)
             print("predicted label",class_names[np.argmax(batch_prediction[0])])
```

```
firt image to be predicted
actual label :  PNEUMONIA
1/1 [==============================] - 0s 249ms/step
predicted label PNEUMONIA
```

A function for predicting

```
In [30]: def predict(model, img):
             img_array  = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
             img_array = tf.expand_dims(img_array,0) # create a batch

             predictions = model.predict(img_array)
             predicted_class = class_names[np.argmax(predictions[0])]
             confidence = round(100*(np.max(predictions[0])),2)
             return predicted_class, confidence
```

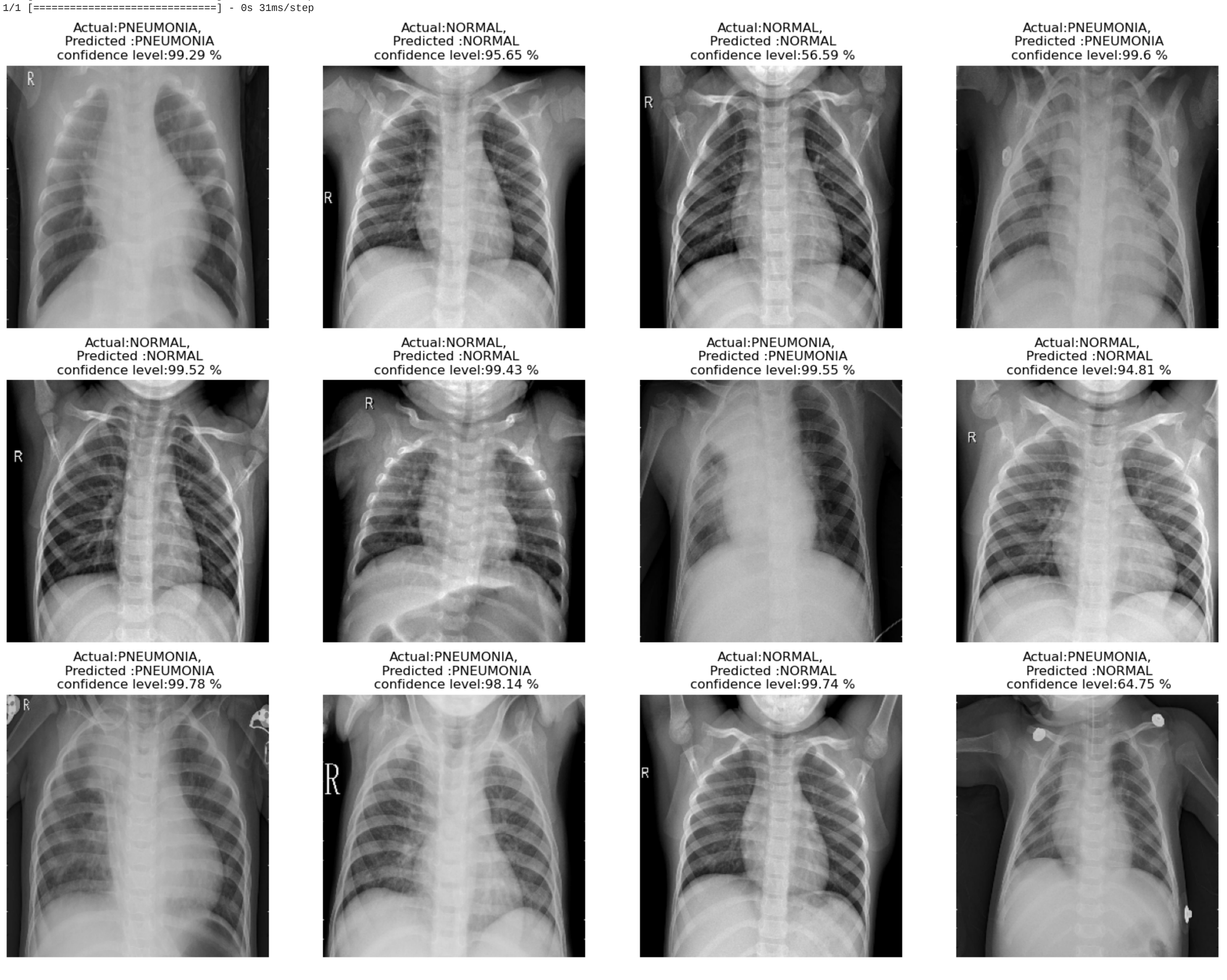
Visualising the predicted images

```
In [34]: plt.figure(figsize=(20,20))

         for images, labels in test_ds.take(1):
             for i in range(12):
                 ax = plt.subplot(4,4,i+1)
                 plt.imshow(images[i].numpy().astype("uint8"))

                 predicted_class, confidence = predict(model, images[i].numpy())
                 actual_class = class_names[labels[i]]
                 plt.title(f"Actual:{actual_class},\n Predicted :{predicted_class}\n confidence level:{confidence} %")
                 plt.axis(False)
```

```
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```



Saving the model

```
In [36]: # Saving the model

         model_version = "pneumonia_prediction_v1"
         model.save(f"/content/drive/MyDrive/models/{model_version}")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/models/pneumonia_prediction_v1\assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/models/pneumonia_prediction_v1\assets
```

```
In [ ]:
```