Identifying Pneumonia Patients based on X-Ray Images

Deanna Gould
Phase 4 Flex Student
Instructor: Morgan Jones
Presentation Date: September 27, 2023

Overview

HealthWorx is a telehealth company that would like to be able to diagnose patients with pneumonia from an X-Ray. X-Ray images can be taken in several locations, and this could decrease wait times for patients. Based on the CDC Website (https://www.cdc.gov/nchs/fastats/pneumonia.htm), 41,309 people die from pneumonia each year, and 1.5 million people visit the emergency room with pneumonia as the primary diagnosis. Emergency rooms are known for their long wait times and becoming overcrowded, so this could also improve other patient's experiences. Pneumonia can have long-lasting effects on the health and well-being of patients. This jupyter notebook will take steps to predict whether a patient has pneumonia or not by using neural networks and image classification of X-Ray images. Although this wouldn't be able to completely replace a doctor's part in diagnosing the patient, this could be used as an added precaution.

The dataset consists of 4,818 images for train data, 418 images for test data, and 624 images for validation data. Different algorithms like will be used and each model will be tuned to determine the best model. Binary cross-entropy will be used as the loss function because this is a binary classification problem. For evaluation metrics, accuracy score, recall, and precision will be considered, but recall will be most important because pneumonia is a health-risk. Recall is the number of true positives divided by the number of true positives and false negatives. A false negative can be detrimental in healthcare settings.

Importing Libraries

In [1]:
```
# Importing libraries

import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (plot_confusion_matrix, confusion_matrix, class
                             RocCurveDisplay)
import tensorflow as tf
from tensorboard.plugins.hparams import api as hp
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras import optimizers
```

Creating Functions

In [2]:
```
# Creating a function called plot history

def plot_history(history):
    acc = history.history['binary_accuracy']
    val_acc = history.history['val_binary_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc))
    plt.plot(epochs, acc, 'pink', label='Training accuracy')
    plt.plot(epochs, val_acc, 'blue', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.figure()
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
    plt.figure()
    plt.show();
```

In [85]:
```
# Creating a function for rocauc
# Plot will have true positives on y and false positive values on X, with t
# being a straight line.

def plot_roc_auc(y_true, y_score):
    fpr, tpr, thresholds = roc_curve(y_true, y_score)

    print('AUC: {}'.format(auc(fpr, tpr)))
    plt.figure(figsize=(10, 8))
    lw = 2
    plt.plot(fpr, tpr, color='blue',
         lw=lw, label='ROC curve')
    plt.plot([0, 1], [0, 1], color='pink', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.yticks([i/20.0 for i in range(21)])
    plt.xticks([i/20.0 for i in range(21)])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic (ROC) Curve')
    plt.legend(loc='lower right');
    plt.show();
```

In [4]:
```
# Creating a function

# Code below from stack overflow
# https://stackoverflow.com/questions/45413712/keras-get-true-labels-y-test

def pred_labels(model, generator):

# Create lists for storing the predictions and labels
# Labels in this case are actual values and predictions are predicted value
    predictions = []
    labels = []

# Get the total number of labels in generator
# (i.e. the length of the dataset where the generator generates batches fro
    n = len(generator.labels)

# Loop over the generator
    for data, label in generator:
    # Make predictions on data using the model. Store the results.
        predictions.extend(model.predict(data, workers = 4).flatten())

    # Store corresponding labels
        labels.extend(label)

    # We have to break out from the generator when we've processed
    # the entire once (otherwise we would end up with duplicates).
        if (len(label) < generator.batch_size) and (len(predictions) == n):
            break
    return labels, predictions
```

In [81]:
```
#Creating a function to plot
def conf_matrix(y_true, y_pred):

    #Converting probabilities to 0 and 1
    y_pred = np.array([round(x) for x in y_pred])

    cm = confusion_matrix(y_true, y_pred)

    #Plotting confusion matrix using heatmap
    fig, ax = plt.subplots(figsize = (8, 6))
    ax = sns.heatmap(cm, annot=True, cmap='flare', fmt='g')

    ax.set_title('Predictions for Pneumonia cases\n\n');
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ');

    ## Ticket labels - List must be in alphabetical order
    ax.xaxis.set_ticklabels(['Normal','Pneumonia'])
    ax.yaxis.set_ticklabels(['Normal','Pneumonia'])

    ## Display the visualization of the Confusion Matrix.
    plt.show();

    #Calculating normalization
    row_sums = cm.sum(axis=1)
    new_matrix = np.round(cm / row_sums[:, np.newaxis], 3)

    #Plotting confusion matrix using heatmap
    fig, ax = plt.subplots(figsize = (8, 6))
    ax = sns.heatmap(new_matrix, annot=True, cmap='flare', fmt='g')

    ax.set_title('Predictions for Pneumonia cases\n\n')
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ');

    ## Ticket labels - List must be in alphabetical order
    ax.xaxis.set_ticklabels(['Normal','Pneumonia'])
    ax.yaxis.set_ticklabels(['Normal','Pneumonia'])

    ## Display the visualization of the Confusion Matrix.
    plt.show();
```

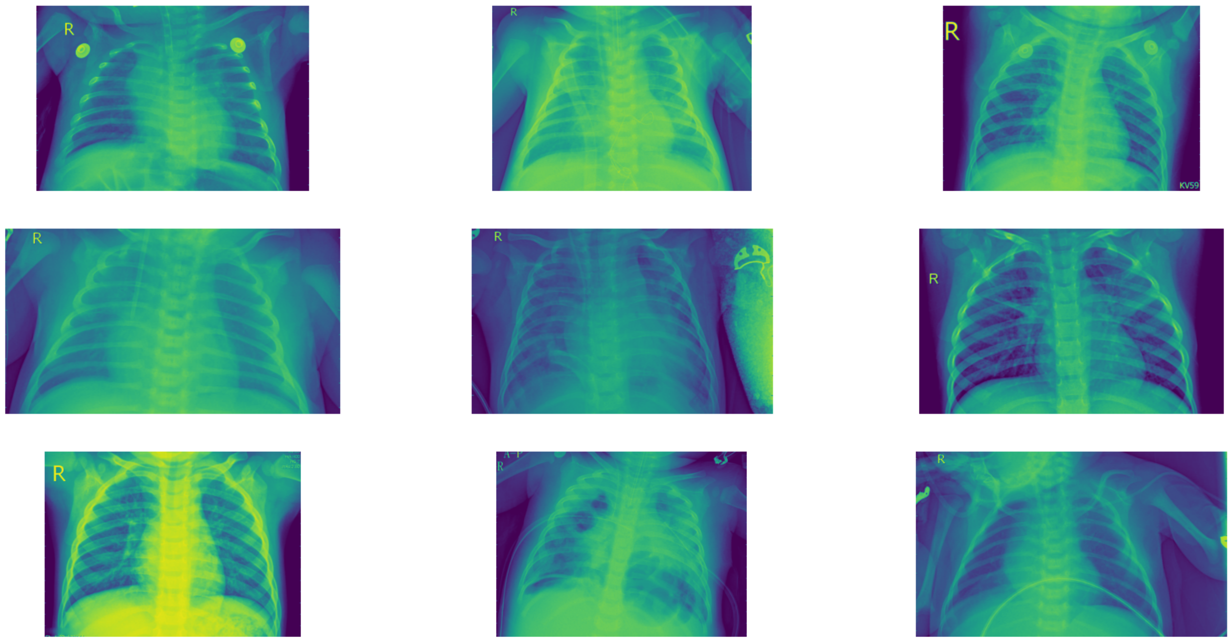
In [6]:
```
# Making directories for train test and validation sets

train_dir = "data/chest_xray/chest_xray/train"
val_dir = "data/chest_xray/chest_xray/val"
test_dir = "data/chest_xray/chest_xray/test"
```

In [7]:
```
# Getting value counts for each directory

print('train_set:')
print('---------')
pneu_count_tr = len(os.listdir(os.path.join(train_dir, 'PNEUMONIA')))
normal_count_tr = len(os.listdir(os.path.join(train_dir, 'NORMAL')))
print(f'Pneumonia = {pneu_count_tr}')
print(f'Normal = {normal_count_tr}')
print('\n')
print('val_set:')
print('---------')
pneu_count_val = len(os.listdir(os.path.join(val_dir, 'PNEUMONIA')))
normal_count_val = len(os.listdir(os.path.join(val_dir, 'NORMAL')))
print(f'Pneumonia = {pneu_count_val}')
print(f'Normal = {normal_count_val}')
print('\n')
print('test_set:')
print('---------')
pneu_count_test = len(os.listdir(os.path.join(test_dir, 'PNEUMONIA')))
normal_count_test = len(os.listdir(os.path.join(test_dir, 'NORMAL')))
print(f'Pneumonia = {pneu_count_test}')
print(f'Normal = {normal_count_test}')
print('\n')
```

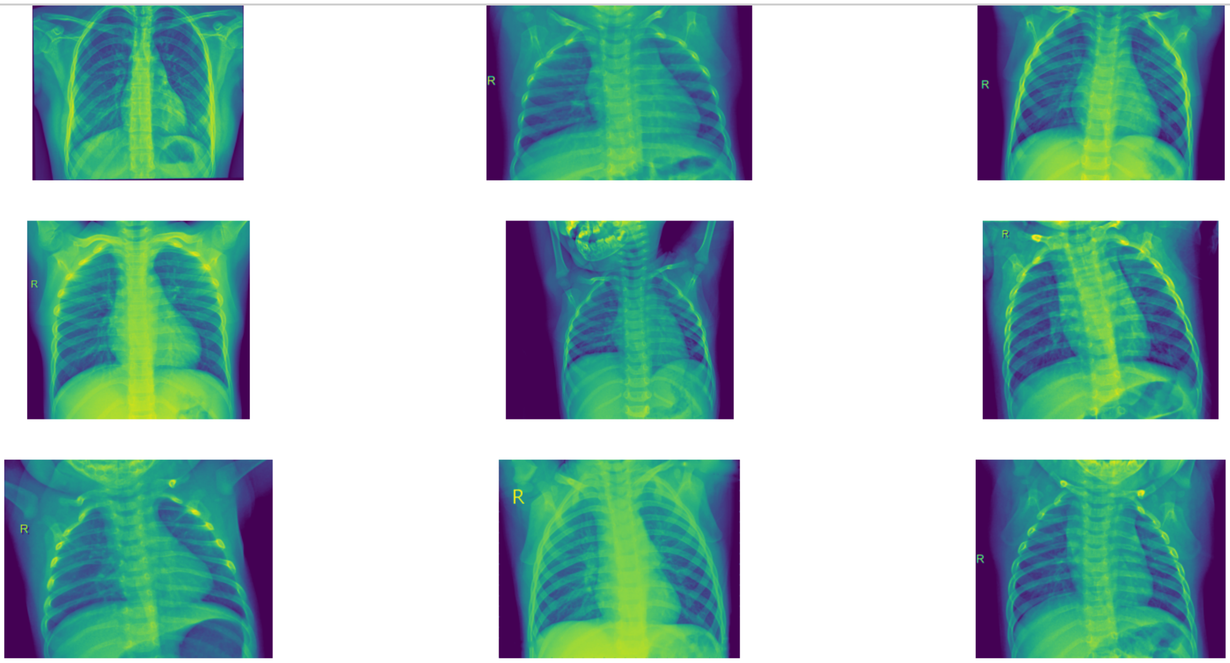
```
train_set:
---------
Pneumonia = 3476
Normal = 942


val_set:
---------
Pneumonia = 409
Normal = 409


test_set:
---------
Pneumonia = 390
Normal = 234
```

It's important to look at the counts of a dataset. Originally, this dataset had only 16 X-Ray images in the validation dataset, so 401 were moved from the train set to the validation set. Still, there are significantly more X-Ray images that show pneumonia than those that don't, which means that the classes need to be weighted.

In [70]:
```
# Displaying pneumonia X-rays
pneumonia = os.listdir("data/chest_xray/chest_xray/train/PNEUMONIA")
pneumoniadir = "data/chest_xray/chest_xray/train/PNEUMONIA"


# Plotting the X-rays
plt.figure(figsize = (20, 10))
for i in range(9):
    plt.subplot(3, 3, i+1)
    image = plt.imread(os.path.join(pneumoniadir, pneumonia[i]))
    plt.imshow(image)
    plt.axis('off')
```



```
In [71]: # Displaying normal X-rays
         normal = os.listdir("data/chest_xray/chest_xray/train/NORMAL")
         normaldir = "data/chest_xray/chest_xray/train/NORMAL"

         plt.figure(figsize = (20, 10))
         for i in range(9):
             plt.subplot(3, 3, i+1)
             image = plt.imread(os.path.join(normaldir, normal[i]))
             plt.imshow(image)
             plt.axis('off')
```



As we can see, the pneumonia images seem to be a little more hazy and less clear, but the X-Ray images are a little difficult to read. Though there aren't many untrained human eyes looking at the X-Rays, it can still be confusing for healthcare providers.

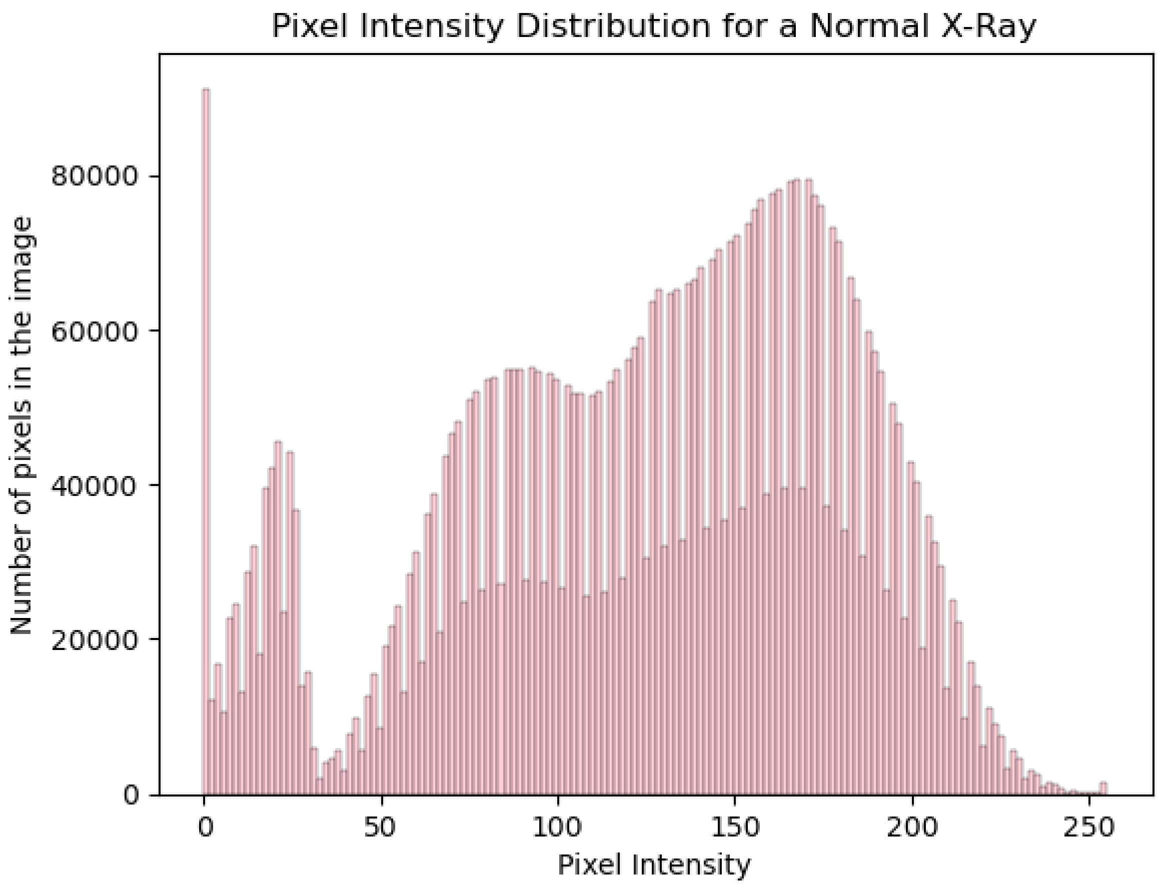
```
In [10]: # Reading the normal images

         normal_img = plt.imread(os.path.join(normaldir, normal[0]))
         normal_img
```

```
Out[10]: array([[ 0, 23, 24, ...,  0,  0,  0],
                [ 0,  5, 23, ...,  0,  0,  0],
                [ 1,  0, 26, ...,  0,  0,  0],
                ...,
                [ 0,  0,  0, ...,  0,  0,  0],
                [ 0,  0,  0, ...,  0,  0,  0],
                [ 0,  0,  0, ...,  0,  0,  0]], dtype=uint8)
```

In [11]:
```
# Reading the pneumonia images

pneumonia_img = plt.imread(os.path.join(pneumoniadir, pneumonia[0]))
pneumonia_img
```

Out[11]:
```
array([[ 0,  0,  0, ..., 47, 46, 45],
       [ 0,  0,  0, ..., 45, 45, 45],
       [ 2,  1,  0, ..., 47, 47, 47],
       ...,
       [ 0,  0,  0, ...,  0,  0,  0],
       [ 0,  0,  0, ...,  0,  0,  0],
       [ 0,  0,  0, ...,  0,  0,  0]], dtype=uint8)
```

In [100]:
```
# Plotting the pixels of the images

sns.histplot(normal_img.ravel(), color = 'pink', bins = 150)
plt.title('Pixel Intensity Distribution for a Normal X-Ray')
plt.xlabel('Pixel Intensity')
plt.ylabel('Number of pixels in the image')
plt.show();
```



In [102]:
```python
# Plotting the pixels of pneumonia images

sns.histplot(pneumonia_img.ravel(), color = 'pink', bins = 150)
plt.title('Pixel Intensity Distribution for a Pneumonia X-Ray')
plt.xlabel('Pixel Intensity')
plt.ylabel('Number of pixels in the image')
sns.histplot(pneumonia_img.ravel(), color = 'pink', bins = 150);
```

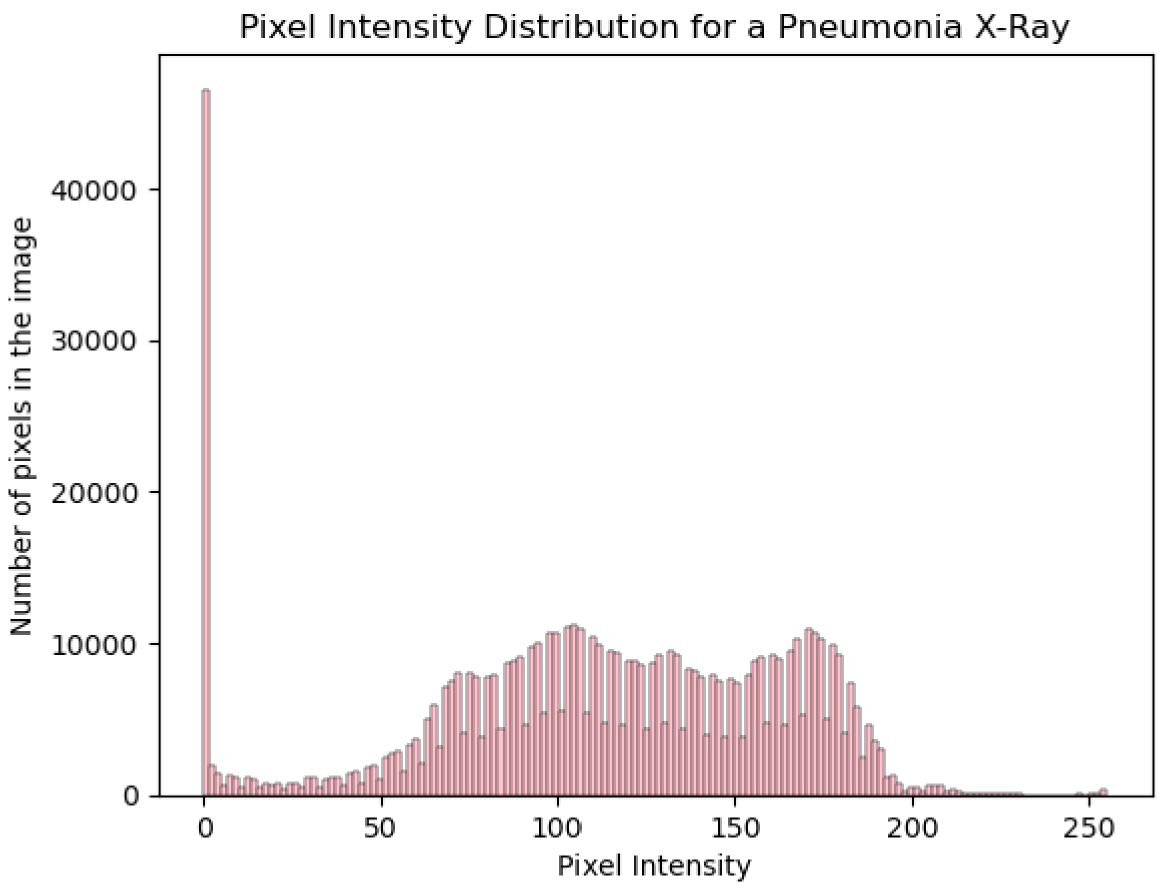


Image Generator

In [14]:
```
# Separating train and val datagen

train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                     target_size=(224, 224),
                                                     batch_size=32,
                                                     class_mode='binary',
                                                     shuffle = True)


validation_generator = val_datagen.flow_from_directory(val_dir,
                                                        target_size=(224, 2
                                                        batch_size=32,
                                                        class_mode='binary'
                                                        shuffle = True)
```

```
Found 4416 images belonging to 2 classes.
Found 816 images belonging to 2 classes.
```

In [15]:
```
# Checking available classes for the validation generator

validation_generator.class_indices
```

Out[15]:
```
{'NORMAL': 0, 'PNEUMONIA': 1}
```

In [16]:
```
# Getting class weights

weight_pneu = pneu_count_tr / (pneu_count_tr + normal_count_tr)

weight_normal = normal_count_tr / (pneu_count_tr + normal_count_tr)

class_weight = {0 : weight_pneu, 1 : weight_normal}
print(f'0 Weight Class = {weight_pneu}')
print(f'1 Weight Class = {weight_normal}')
```

```
0 Weight Class = 0.7867813490267089
1 Weight Class = 0.21321865097329107
```

Baseline Model

In [17]:
```
#Initiating the model
modelone = models.Sequential()

# Input layer
modelone.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))
modelone.add(layers.MaxPooling2D((2, 2)))

#Hidden Layer
modelone.add(layers.Flatten())
modelone.add(layers.Dense(64, activation='relu'))

#Output Layer
modelone.add(layers.Dense(1, activation='sigmoid'))

modelone.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=1e-4),
              metrics=tf.keras.metrics.BinaryAccuracy(name="binary_accuracy
                )
```

```
2023-09-25 17:08:12.228598: I tensorflow/core/platform/cpu_feature_guard.
cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Netwo
rk Library (oneDNN)to use the following CPU instructions in performance-c
ritical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropria
te compiler flags.
2023-09-25 17:08:12.271425: I tensorflow/compiler/xla/service/service.cc:
168] XLA service 0x7fa2eeb41580 initialized for platform Host (this does
not guarantee that XLA will be used). Devices:
2023-09-25 17:08:12.271442: I tensorflow/compiler/xla/service/service.cc:
176]   StreamExecutor device (0): Host, Default Version
```

Only thing that will change is optimizer

In [18]:
```
# Getting summary for model one

modelone.summary()
```

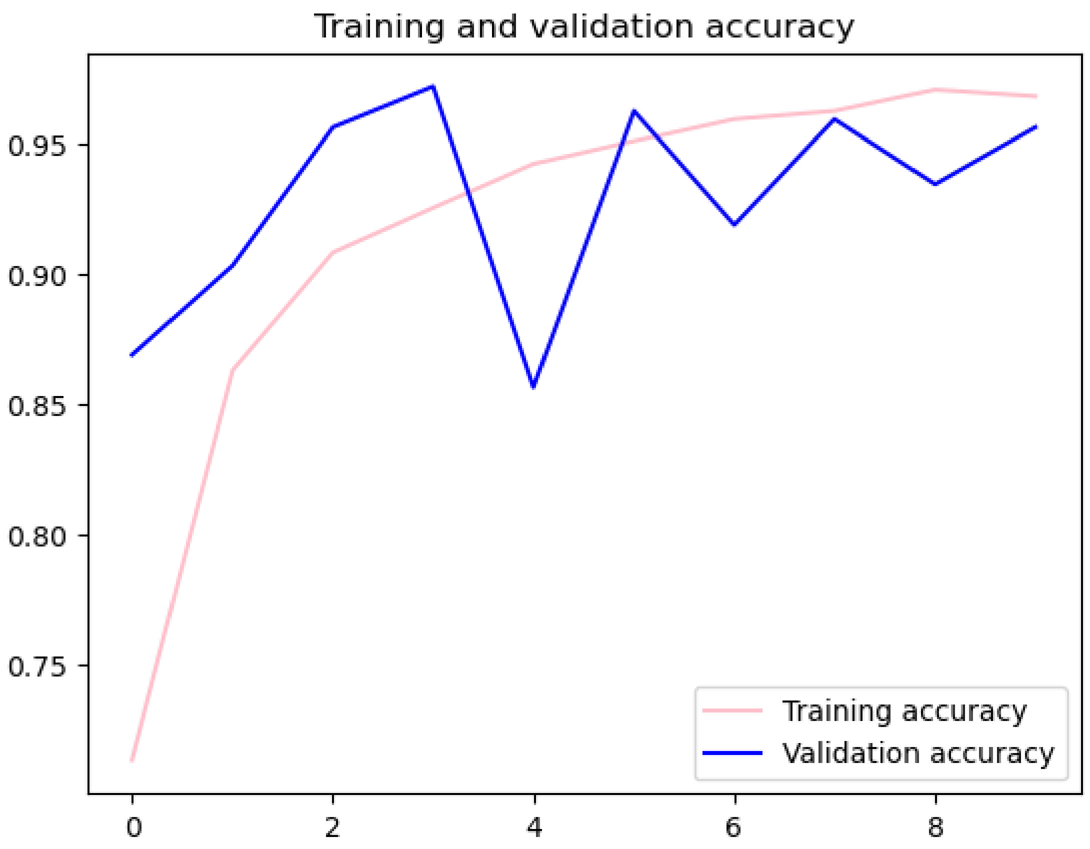
```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)     0
_________________________________________________________________
flatten (Flatten)           (None, 394272)            0
_________________________________________________________________
dense (Dense)               (None, 64)                25233472
_________________________________________________________________
dense_1 (Dense)             (None, 1)                 65
=================================================================
Total params: 25,234,433
Trainable params: 25,234,433
Non-trainable params: 0
_________________________________________________________________
```

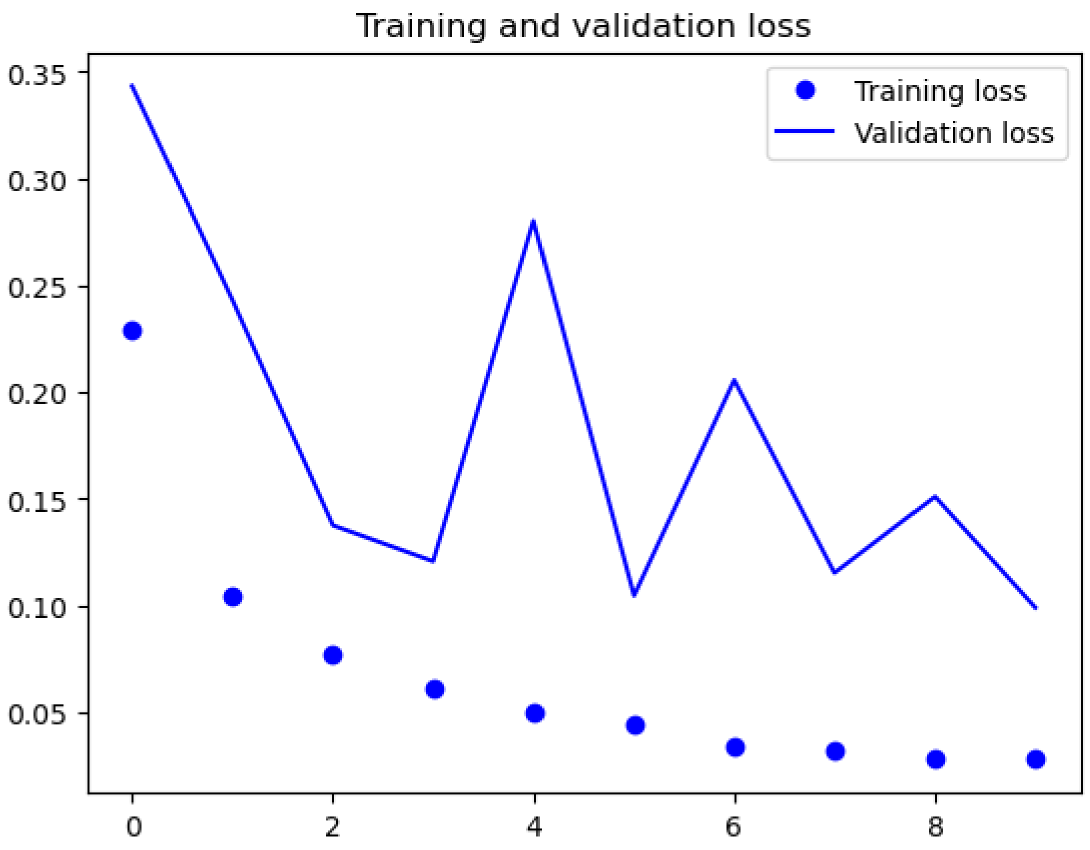
In [19]:
```
# Fitting the first model

historyone = modelone.fit(train_generator,
                              epochs=10,
                              validation_data=validation_generator,
                              class_weight = class_weight,
                              steps_per_epoch = 100,
                              validation_steps=10)
```

```
Epoch 1/10
100/100 [==============================] - 81s 807ms/step - loss: 0.2291
- binary_accuracy: 0.7131 - val_loss: 0.3434 - val_binary_accuracy: 0.868
7
Epoch 2/10
100/100 [==============================] - 82s 820ms/step - loss: 0.1047
- binary_accuracy: 0.8628 - val_loss: 0.2430 - val_binary_accuracy: 0.903
1
Epoch 3/10
100/100 [==============================] - 85s 847ms/step - loss: 0.0773
- binary_accuracy: 0.9081 - val_loss: 0.1376 - val_binary_accuracy: 0.956
3
Epoch 4/10
100/100 [==============================] - 86s 860ms/step - loss: 0.0611
- binary_accuracy: 0.9253 - val_loss: 0.1208 - val_binary_accuracy: 0.971
9
Epoch 5/10
100/100 [==============================] - 85s 854ms/step - loss: 0.0504
- binary_accuracy: 0.9422 - val_loss: 0.2804 - val_binary_accuracy: 0.856
2
Epoch 6/10
100/100 [==============================] - 86s 857ms/step - loss: 0.0442
- binary_accuracy: 0.9509 - val_loss: 0.1046 - val_binary_accuracy: 0.962
5
Epoch 7/10
100/100 [==============================] - 86s 858ms/step - loss: 0.0345
- binary_accuracy: 0.9594 - val_loss: 0.2058 - val_binary_accuracy: 0.918
7
Epoch 8/10
100/100 [==============================] - 86s 860ms/step - loss: 0.0319
- binary_accuracy: 0.9625 - val_loss: 0.1153 - val_binary_accuracy: 0.959
4
Epoch 9/10
100/100 [==============================] - 86s 862ms/step - loss: 0.0281
- binary_accuracy: 0.9706 - val_loss: 0.1511 - val_binary_accuracy: 0.934
4
Epoch 10/10
100/100 [==============================] - 86s 862ms/step - loss: 0.0280
- binary_accuracy: 0.9681 - val_loss: 0.0990 - val_binary_accuracy: 0.956
3
```

Baseline Training Validation Accuracy

In [86]:
```
# Plotting history for model one

plot_history(historyone)
```





```
<Figure size 640x480 with 0 Axes>
```

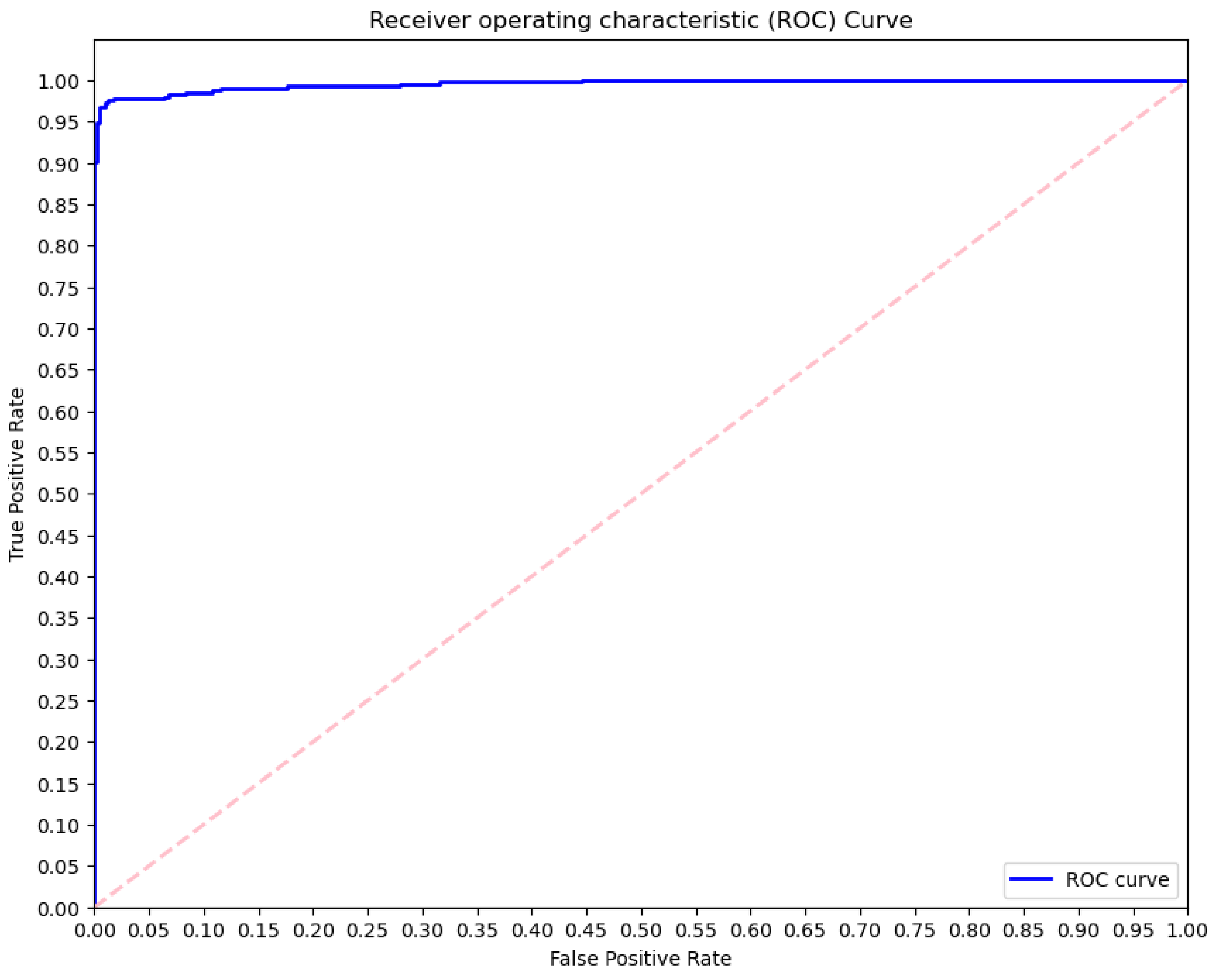
In this first baseline model, the first graph which plots training and validation accuracy. The model is fitting to the training set much better than the validation set. The validation accuracy line is muore jagged and inconsistent than the training set. When looking at loss, the training loss is more jagged and inconsistent, and the width between the two plots are more than I would hope.

Baseline Predictions Check

```
In [22]: # Calling pred_labels to see performance of model 1

         modelone_predsval = pred_labels(modelone, validation_generator)
```

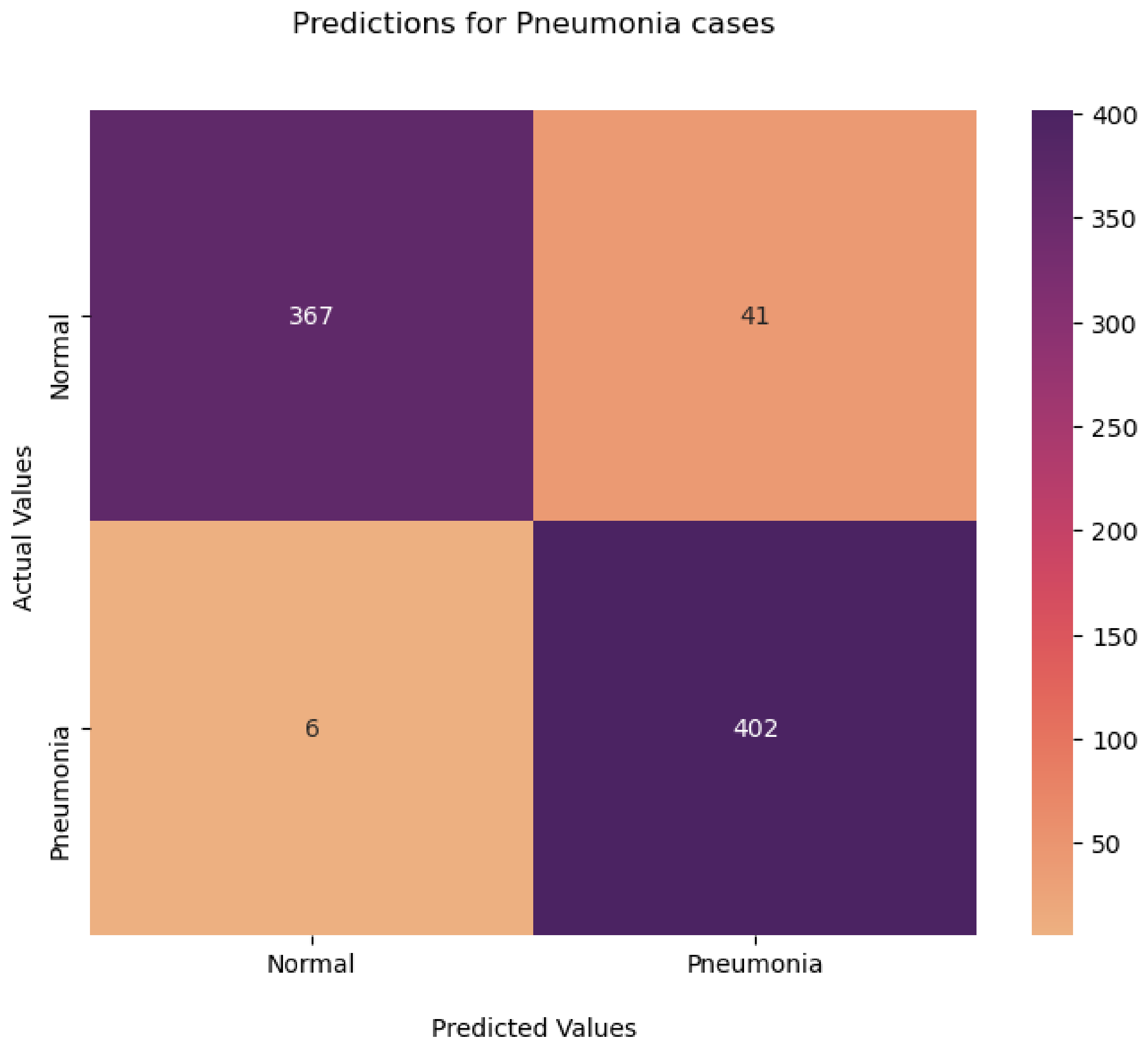
```
In [24]: # Plotting the performance of model 1

         plot_roc_auc(modelone_predsval[0], modelone_predsval[1])
```

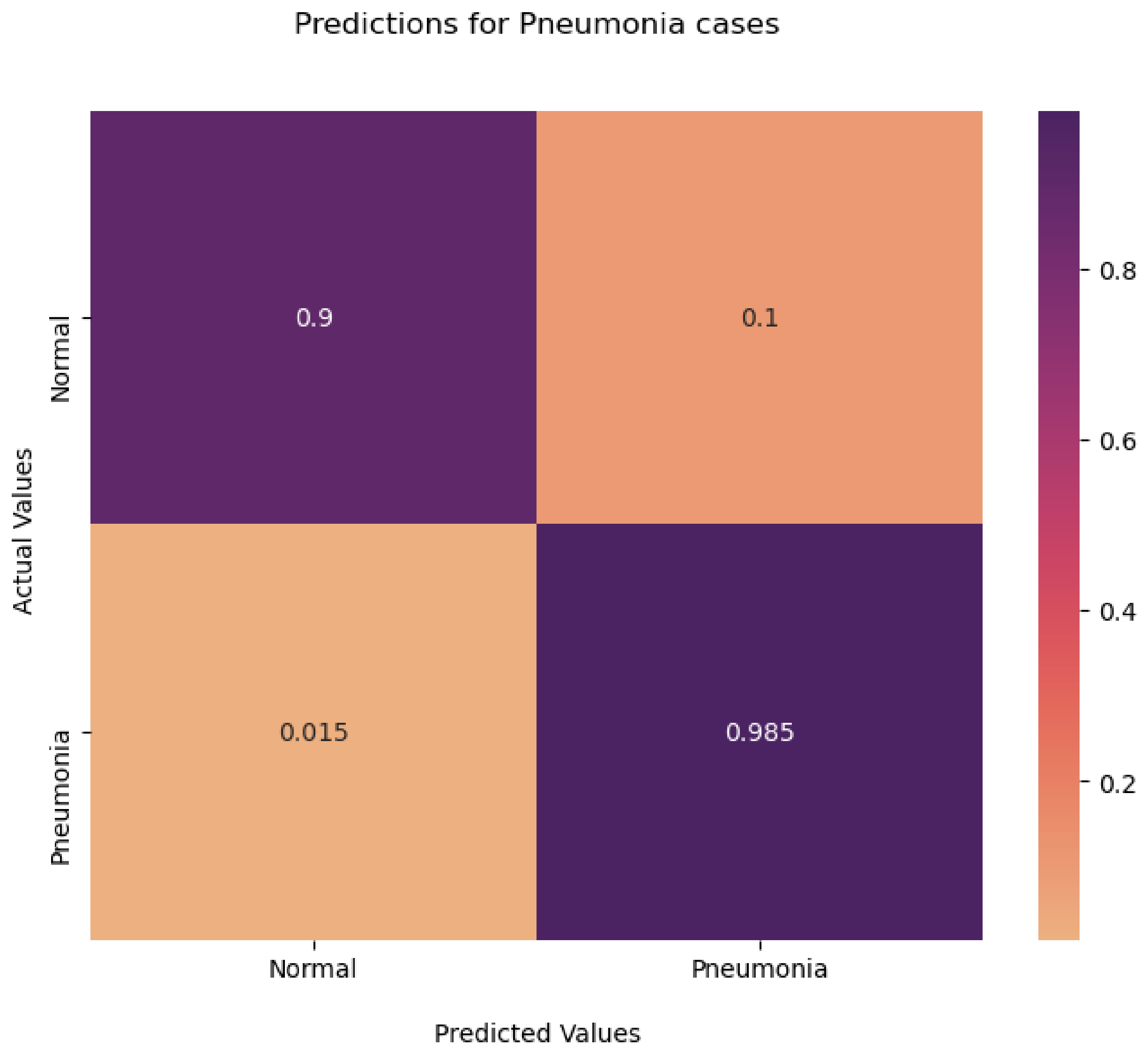
```
AUC: 0.995608660130719
```



The ROC curve above shows that the model learned incredibly fast with a true positive rate. The ROC curve peaks soon after some minor growth, but as time goes on, the false positive rate will continue to increase while the true positive plateaus.

In [82]:
```
# Printing confusion matrix for validation set

conf_matrix(modelone_predsval[0], modelone_predsval[1])
```





This confusion matrix of the baseline model shows that there are 98.5% true positives, and 1.5% false negatives, which represents the amount of patients not getting diagnosed. This is okay, but not reliable enough for this use case. Out of the patients with normal X-Ray images, 10% are false positives. It is better that the percentage of false positives is higher than the percentage of false negatives.

HParams

In [27]:
```
# Creating the hparam variables

HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([64, 128]))
HP_DROPOUT = hp.HParam('dropout_rate', hp.RealInterval(0.1, 0.2))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'rmsprop']))
HP_LEARNING_RATE = hp.HParam('learning_rate', hp.Discrete([0.01, 0.001, 0.0
METRIC_ACCURACY = 'binary_accuracy'
```

In [28]:
```
#Creating a directory for logs

logdir = 'logs/hparam_tuning'
```

In [30]:
```
# Creating a file writer for hyperparameter tuning

with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
  hp.hparams_config(
    hparams=[HP_NUM_UNITS, HP_DROPOUT, HP_OPTIMIZER, HP_LEARNING_RATE],
    metrics=[hp.Metric(METRIC_ACCURACY, display_name='binary_accuracy')],
  )
```

In [75]:
```
# The function below uses the baseline model as it's base model. It changes
# learning rate based on the set params in the HParams.

# Creating function to do an HParams search
def create_model_grid(hparams):
    #Initializing model
    model = models.Sequential()

    #Adding CNN input layer
    model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (
    model.add(layers.MaxPooling2D(2,2))
    model.add(layers.Dropout(hparams[HP_DROPOUT]))

    #Adding Dense hidden layer
    model.add(layers.Flatten())
    model.add(layers.Dense(hparams[HP_NUM_UNITS], activation = 'relu'))
    model.add(layers.Dropout(hparams[HP_DROPOUT]))

    #Adding output layer
    model.add(layers.Dense(1, activation = 'sigmoid'))

    #Looping through optimizers and learning rates
    optimizer = hparams[HP_OPTIMIZER]
    learning_rate = hparams[HP_LEARNING_RATE]
    if optimizer == "adam":
        optimizer = tf.optimizers.Adam(learning_rate=learning_rate)
    elif optimizer=='rmsprop':
        optimizer = tf.optimizers.RMSprop(learning_rate=learning_rate)
    else:
        raise ValueError("unexpected optimizer name: %r" % (optimizer_name,

    #Compiling model
    model.compile(loss= 'binary_crossentropy',
    optimizer= optimizer,
    metrics= tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", dtype=

    #Fitting model
    history=model.fit(
    train_generator, #Using train data
    steps_per_epoch=100, #Keeping 100 steps
    epochs=10, #Keeping 10 epochs
    validation_data=validation_generator, #Using validation data
    class_weight = class_weight, #Adding weights to deal with imbalance
    validation_steps=10, #Keeping 10 steps
    )

    return history.history['val_binary_accuracy'][-1]
```

In [76]:
```python
# Creating run function

def run(run_dir, hparams):
  with tf.summary.create_file_writer(run_dir).as_default():
    hp.hparams(hparams)  # record the values used in this trial
    accuracy = create_model_grid(hparams)
    tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

In [33]:
```python
# Running hparams model

session_num = 0

for num_units in HP_NUM_UNITS.domain.values:
  for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_v
    for optimizer in HP_OPTIMIZER.domain.values:
        for learning_rate in HP_LEARNING_RATE.domain.values:
            hparams = {
              HP_NUM_UNITS: num_units,
              HP_DROPOUT: dropout_rate,
              HP_OPTIMIZER: optimizer,
              HP_LEARNING_RATE: learning_rate
              }
            run_name = "run-%d" % session_num
            print('--- Starting trial: %s' % run_name)
            print({h.name: hparams[h] for h in hparams})
            run('logs/hparam_tuning/' + run_name, hparams)
            session_num += 1
```

```
3
Epoch 6/10
100/100 [==============================] - 89s 889ms/step - loss: 0.0592
- binary_accuracy: 0.9359 - val_loss: 0.1483 - val_binary_accuracy: 0.950
0
Epoch 7/10

100/100 [==============================] - 87s 871ms/step - loss: 0.0525
- binary_accuracy: 0.9441 - val_loss: 0.1183 - val_binary_accuracy: 0.953
1
Epoch 8/10
100/100 [==============================] - 88s 879ms/step - loss: 0.0512
- binary_accuracy: 0.9478 - val_loss: 0.1365 - val_binary_accuracy: 0.950
0
Epoch 9/10
100/100 [==============================] - 86s 864ms/step - loss: 0.0435
- binary_accuracy: 0.9547 - val_loss: 0.1091 - val_binary_accuracy: 0.959
4
Epoch 10/10
100/100 [==============================] - 88s 876ms/step - loss: 0.0384
```

Here are some of the best parameters below.

run-4
{'num_units': 64, 'dropout_rate': 0.1, 'optimizer': 'rmsprop', 'learning_rate': 0.001}
84s 839ms/step - loss: 0.0478 - binary_accuracy: 0.9681 - val_loss: 0.1161 - val_binary_accuracy: 0.9625

run-6
{'num_units': 64, 'dropout_rate': 0.2, 'optimizer': 'adam', 'learning_rate': 0.0001}
79s 790ms/step - loss: 0.0243 - binary_accuracy: 0.9759 - val_loss: 0.0816 - val_binary_accuracy: 0.9750

run-21
{'num_units': 128, 'dropout_rate': 0.2, 'optimizer': 'rmsprop', 'learning_rate': 0.0001}
89s 889ms/step - loss: 0.0358 - binary_accuracy: 0.9625 - val_loss: 0.0964 - val_binary_accuracy: 0.9750

The best runs of the model are above. The best optimizer is adam, and the best dropout rate is 0.2. Run 6 is the best parameters out of the 3, because the binary accuracy is highest of 0.9759, and the binary accuracy of the validation set is *very* close at 0.9750. This means that the model is not overfitting, and is performing well with train *and* validation data. Now that we know which run has the best parameters, those parameters are what I will use to tune the model.

Tuning HParams

In [34]:
```
# Instantiating model 2

model2 = models.Sequential()

#Adding CNN input layer
model2.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (224
model2.add(layers.MaxPooling2D(2,2))

# Adding 0.2 to .Dropout since that was the best dropout parameter
model2.add(layers.Dropout(0.2))

#Adding Dense hidden layer
model2.add(layers.Flatten())

# Adding best num_units to dense layer
model2.add(layers.Dense(64, activation = 'relu'))
model2.add(layers.Dropout(0.2))

#Adding output layer
model2.add(layers.Dense(1, activation = 'sigmoid'))

#Looping through optimizers and learning rates

#Compiling model
model2.compile(loss= 'binary_crossentropy',
optimizer= optimizers.Adam(lr = 1e-4),
metrics= tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", dtype=None
```

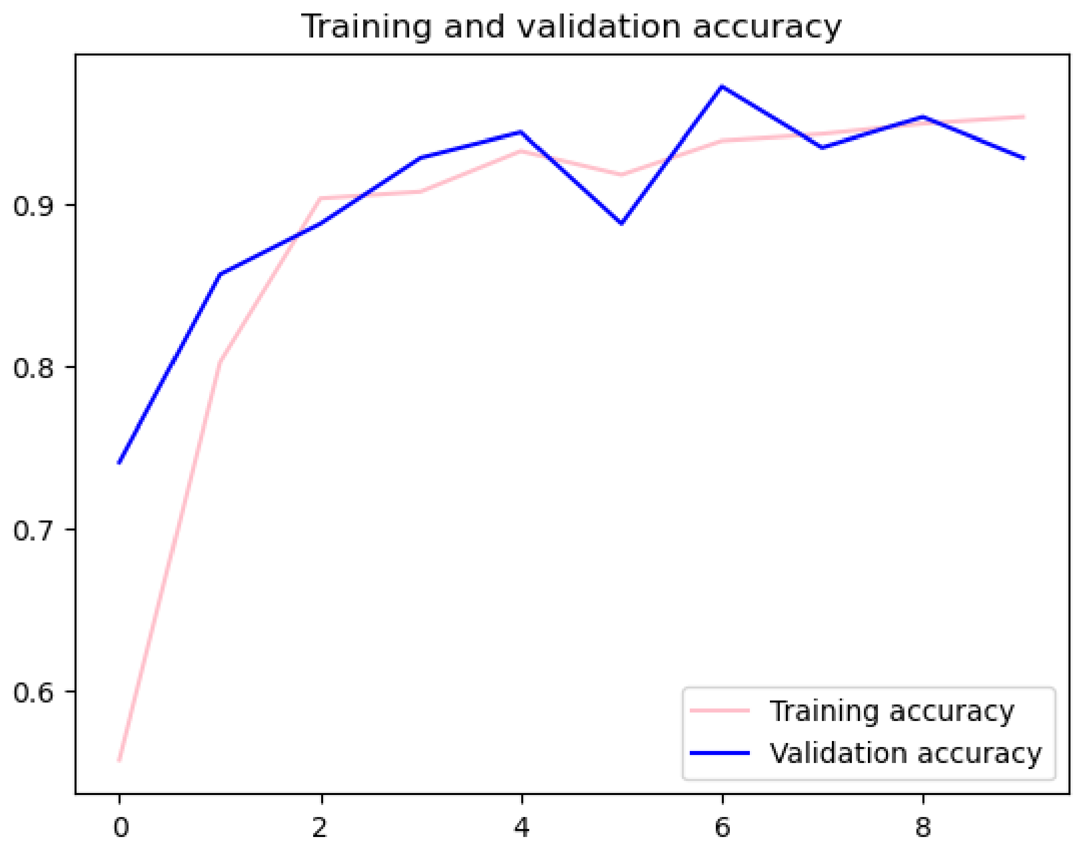
In [35]:
```
# Printing the summary for model 2

model2.summary()
```

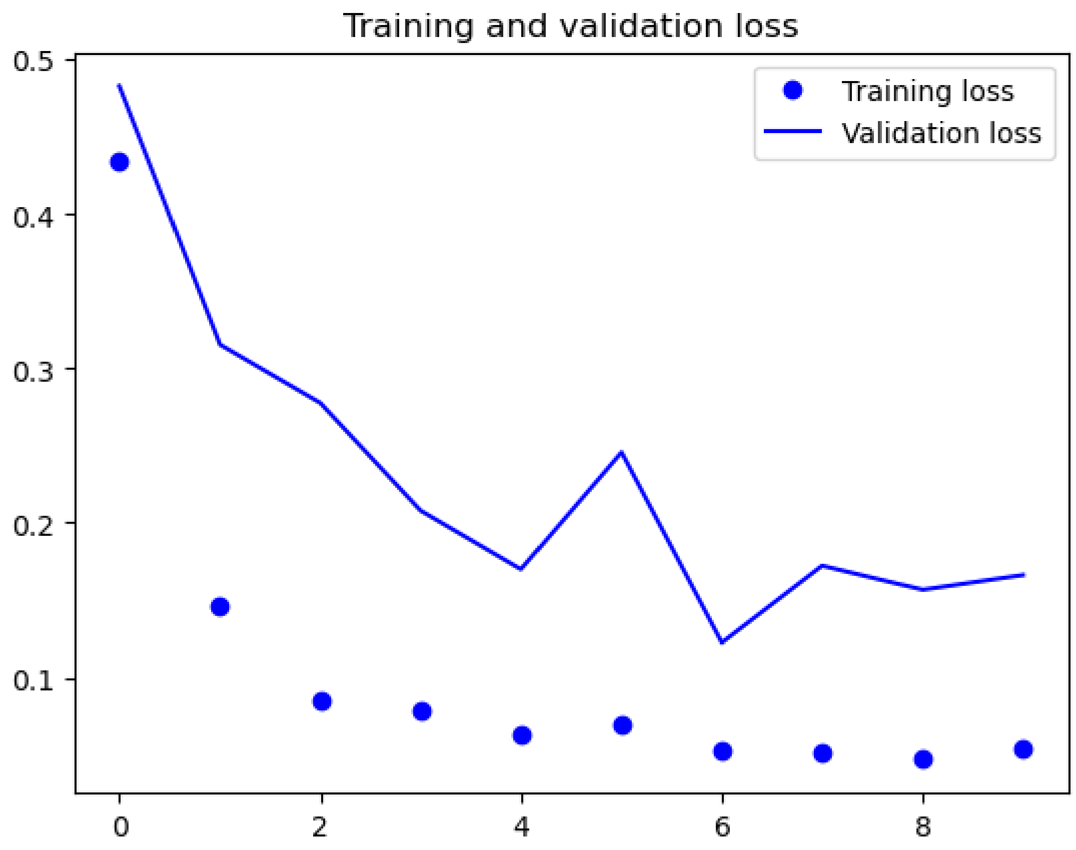
```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 111, 111, 32)      0
_________________________________________________________________
dropout_48 (Dropout)         (None, 111, 111, 32)      0
_________________________________________________________________
flatten_25 (Flatten)         (None, 394272)            0
_________________________________________________________________
dense_50 (Dense)             (None, 64)                25233472
_________________________________________________________________
dropout_49 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_51 (Dense)             (None, 1)                 65
=================================================================
Total params: 25,234,433
Trainable params: 25,234,433
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
#Fitting model 2

history_hparams=model2.fit(
train_generator, #Using train data
steps_per_epoch=30, #Keeping 30 steps
epochs=10, #Keeping 10 epochs
validation_data=validation_generator, #Using validation data
class_weight = class_weight, #Adding weights to deal with imbalance
validation_steps=10, #Keeping 10 steps
)
```

```
30/30 [==============================] - 28s 929ms/step - loss: 0.0782 -
binary_accuracy: 0.9073 - val_loss: 0.2079 - val_binary_accuracy: 0.9281
Epoch 5/10
30/30 [==============================] - 30s 1s/step - loss: 0.0637 - bin
ary_accuracy: 0.9323 - val_loss: 0.1699 - val_binary_accuracy: 0.9438
Epoch 6/10
30/30 [==============================] - 27s 914ms/step - loss: 0.0701 -
binary_accuracy: 0.9177 - val_loss: 0.2457 - val_binary_accuracy: 0.8875
Epoch 7/10
30/30 [==============================] - 27s 903ms/step - loss: 0.0528 -
binary_accuracy: 0.9385 - val_loss: 0.1224 - val_binary_accuracy: 0.9719
Epoch 8/10
30/30 [==============================] - 27s 895ms/step - loss: 0.0519 -
binary_accuracy: 0.9427 - val_loss: 0.1722 - val_binary_accuracy: 0.9344
Epoch 9/10
30/30 [==============================] - 28s 935ms/step - loss: 0.0469 -
binary_accuracy: 0.9490 - val_loss: 0.1566 - val_binary_accuracy: 0.9531
Epoch 10/10
30/30 [==============================] - 27s 908ms/step - loss: 0.0534 -
binary_accuracy: 0.9531 - val_loss: 0.1661 - val_binary_accuracy: 0.9281
```

In [37]:
```
# Calling the plot history function created earlier

plot_history(history_hparams)
```





```
<Figure size 640x480 with 0 Axes>
```

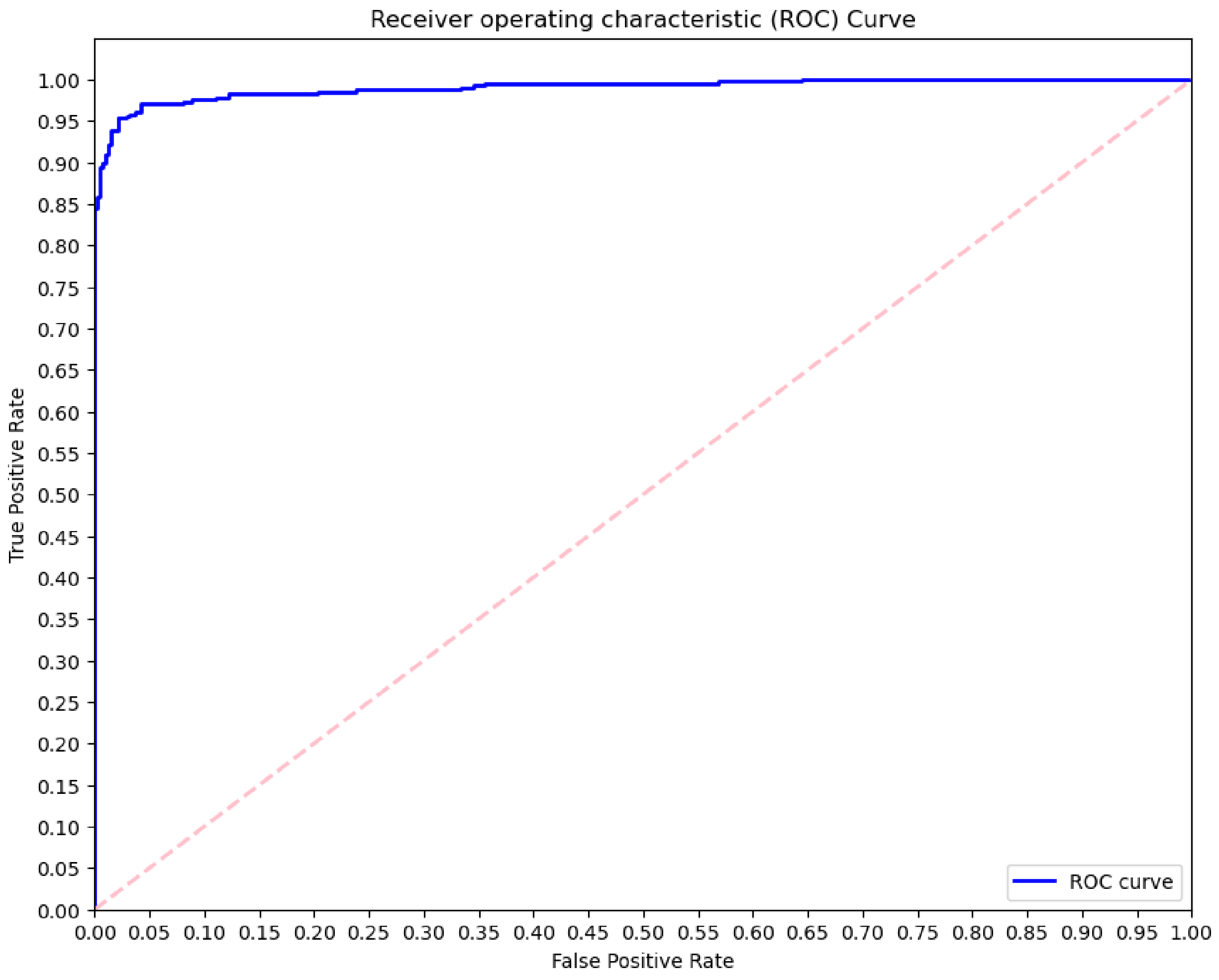
In the top graph above, the validation and training accuracy are plotted. The training accuracy curve is a much smoother curve than the validation curve, which means this second model is performing better on the validation set, but still not optimal performance. In the bottom graph that's above, the training loss and validation loss are plotted. The training loss shows how well the model is fitting the training data, and the validation loss shows how well the model fits new data. That being said, there is room to improve, so I'm going to check the predictions to determine next steps.

HParams Predictions Check

```
In [39]: model2_predsval = pred_labels(model2, validation_generator)
```

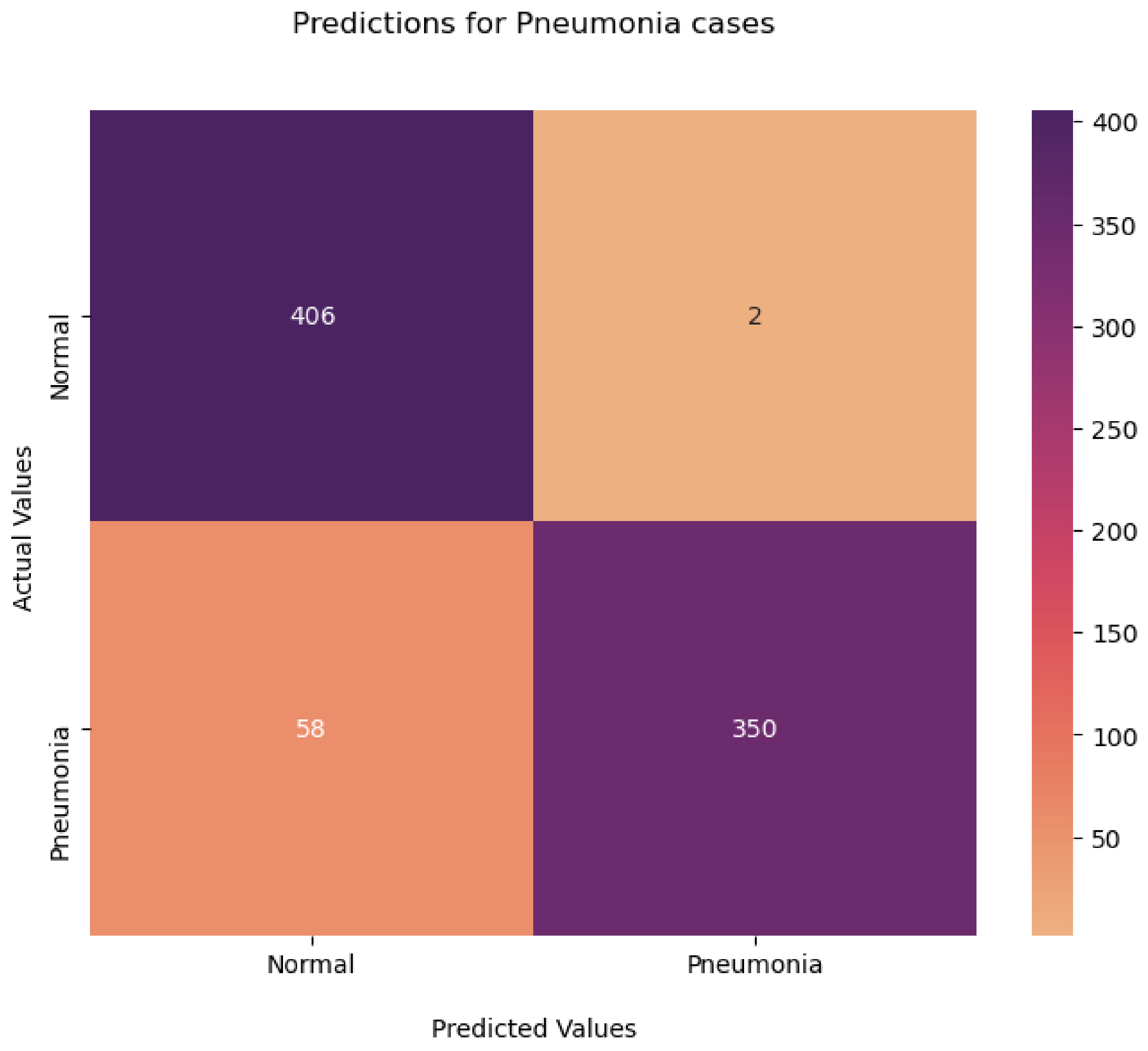
```
In [41]: plot_roc_auc(model2_predsval[0], model2_predsval[1])
```

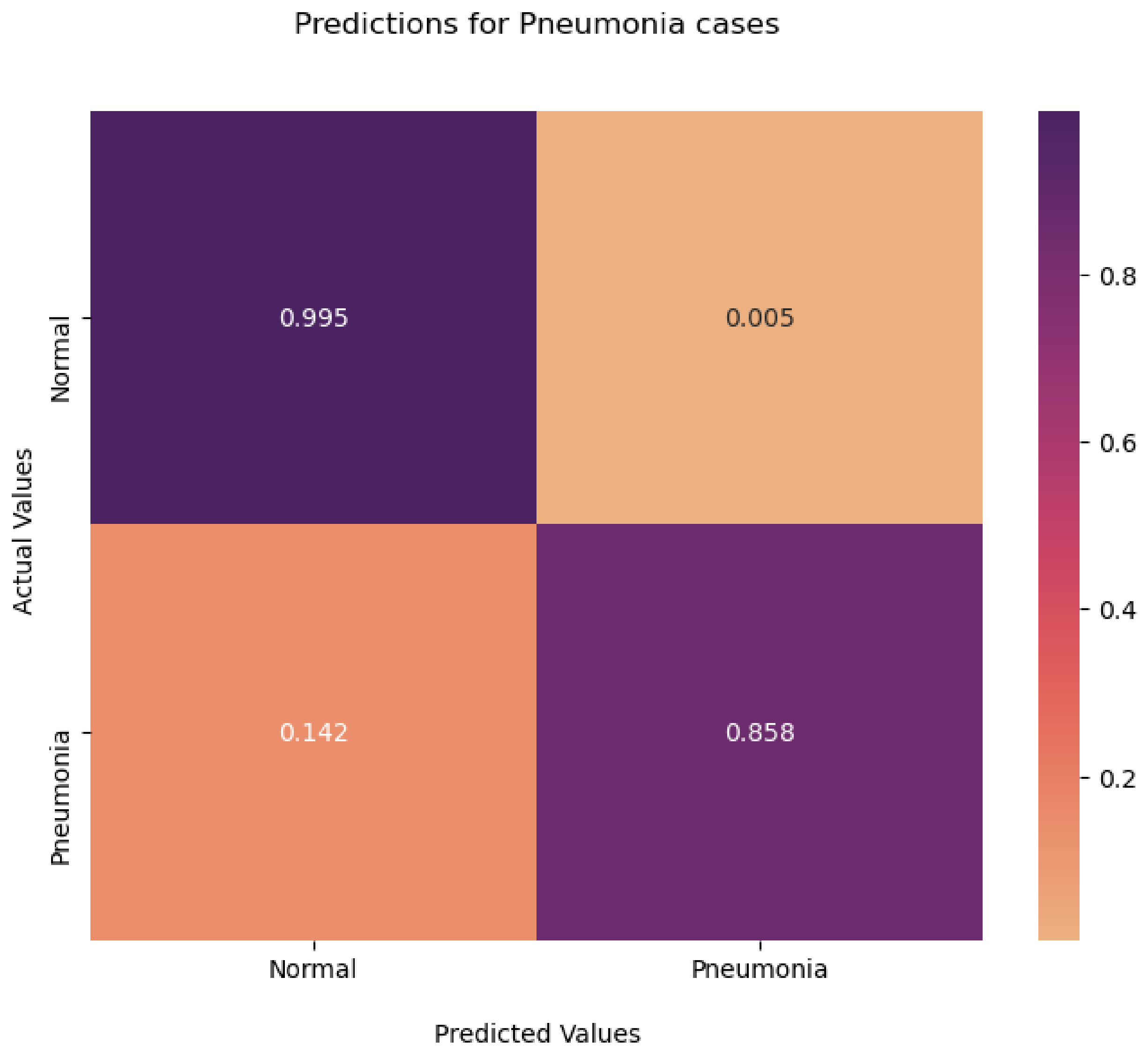
```
AUC: 0.9904063341022683
```



The ROC/AUC curve of model 2 shows similar results to the baseline model, but did grow slightly more beofre reaching a plateau.

In [87]:
```
conf_matrix(model2_predsval[0], model2_predsval[1])
```





The confusion matrices above aren't showing the results we would like. False negatives and false positives seem to have switched places since the baseline model. 14.2% of patients with pneumonia are predicted to not have it, but the false positive rate is much lower than the. baseline at 0.5%. However, like I mentioned before, in healthcare cases, it's best to have more false positives than it is false negatives.

Transfer Learning

Inception-V3

In [44]:
```
# Instantiating model 3

model3 = models.Sequential()

# Creating an inception v3 model

inception_v3 = tf.keras.applications.InceptionV3(
    include_top=False,
    weights="imagenet",
    input_shape=(224, 224, 3),
    classes=1
)

for layer in inception_v3.layers:
    layer.trainable = False

model3.add(inception_v3)
```

In [45]:
```
model3.add(layers.GlobalAveragePooling2D())
model3.add(layers.Dense(64, activation = 'relu'))
model3.add(layers.Dropout(0.2))

#Adding output layer
model3.add(layers.Dense(1, activation = 'sigmoid'))
```

In [46]:
```
# Compiling model
model3.compile(loss= 'binary_crossentropy',
optimizer= optimizers.Adam(lr = 1e-4),
metrics= tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", dtype=None
```

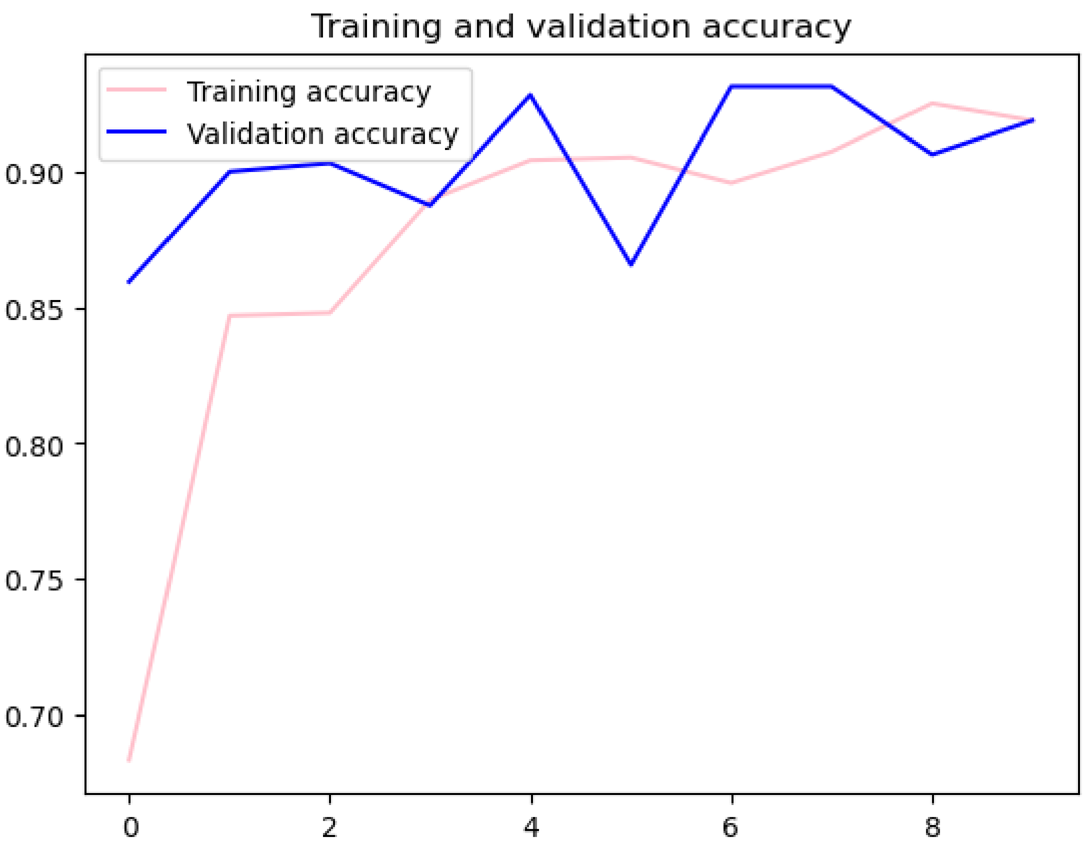
In [47]:
```
model3.summary()
```

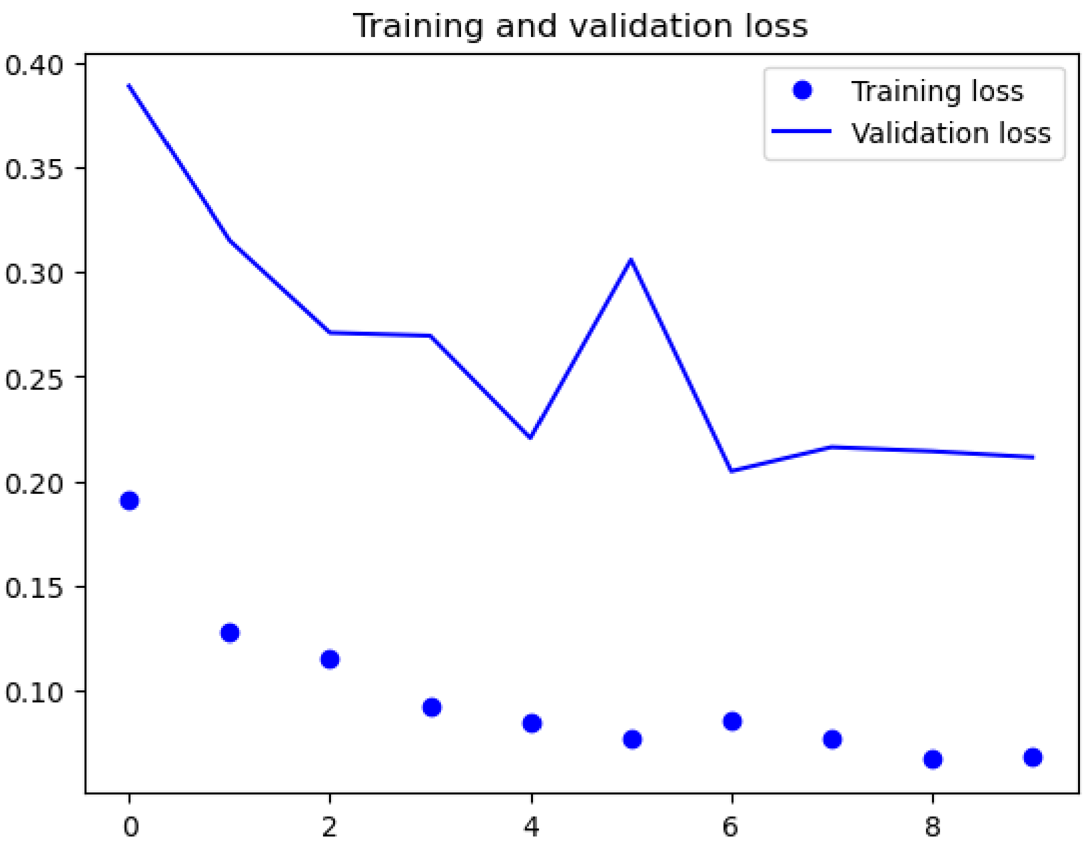
```
Model: "sequential_26"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 5, 5, 2048)        21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense_52 (Dense)             (None, 64)                131136
_________________________________________________________________
dropout_50 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_53 (Dense)             (None, 1)                 65
=================================================================
Total params: 21,933,985
Trainable params: 131,201
Non-trainable params: 21,802,784
_________________________________________________________________
```

In [48]:
```
#Fitting model 3

history_transfer=model3.fit(
train_generator, #Using train data
steps_per_epoch=30, #Keeping 30 steps
epochs=10, #Keeping 10 epochs
validation_data=validation_generator, #Using validation data
class_weight = class_weight, #Adding weights to deal with imbalance
validation_steps=10, #Keeping 10 steps
)
```

```
Epoch 1/10
30/30 [==============================] - 29s 958ms/step - loss: 0.1907 -
binary_accuracy: 0.6833 - val_loss: 0.3890 - val_binary_accuracy: 0.8594
Epoch 2/10
30/30 [==============================] - 30s 991ms/step - loss: 0.1278 -
binary_accuracy: 0.8469 - val_loss: 0.3152 - val_binary_accuracy: 0.9000
Epoch 3/10
30/30 [==============================] - 35s 1s/step - loss: 0.1157 - bin
ary_accuracy: 0.8479 - val_loss: 0.2711 - val_binary_accuracy: 0.9031
Epoch 4/10
30/30 [==============================] - 32s 1s/step - loss: 0.0924 - bin
ary_accuracy: 0.8896 - val_loss: 0.2696 - val_binary_accuracy: 0.8875
Epoch 5/10
30/30 [==============================] - 34s 1s/step - loss: 0.0845 - bin
ary_accuracy: 0.9042 - val_loss: 0.2206 - val_binary_accuracy: 0.9281
Epoch 6/10
30/30 [==============================] - 33s 1s/step - loss: 0.0774 - bin
ary_accuracy: 0.9052 - val_loss: 0.3060 - val_binary_accuracy: 0.8656
Epoch 7/10
30/30 [==============================] - 33s 1s/step - loss: 0.0858 - bin
ary_accuracy: 0.8958 - val_loss: 0.2048 - val_binary_accuracy: 0.9312
Epoch 8/10
30/30 [==============================] - 33s 1s/step - loss: 0.0773 - bin
ary_accuracy: 0.9073 - val_loss: 0.2163 - val_binary_accuracy: 0.9312
Epoch 9/10
30/30 [==============================] - 34s 1s/step - loss: 0.0670 - bin
ary_accuracy: 0.9250 - val_loss: 0.2143 - val_binary_accuracy: 0.9062
Epoch 10/10
30/30 [==============================] - 33s 1s/step - loss: 0.0688 - bin
ary_accuracy: 0.9187 - val_loss: 0.2115 - val_binary_accuracy: 0.9187
```

In [49]:
```
# Calling the plot history function for the transfer tuning model

plot_history(history_transfer)
```





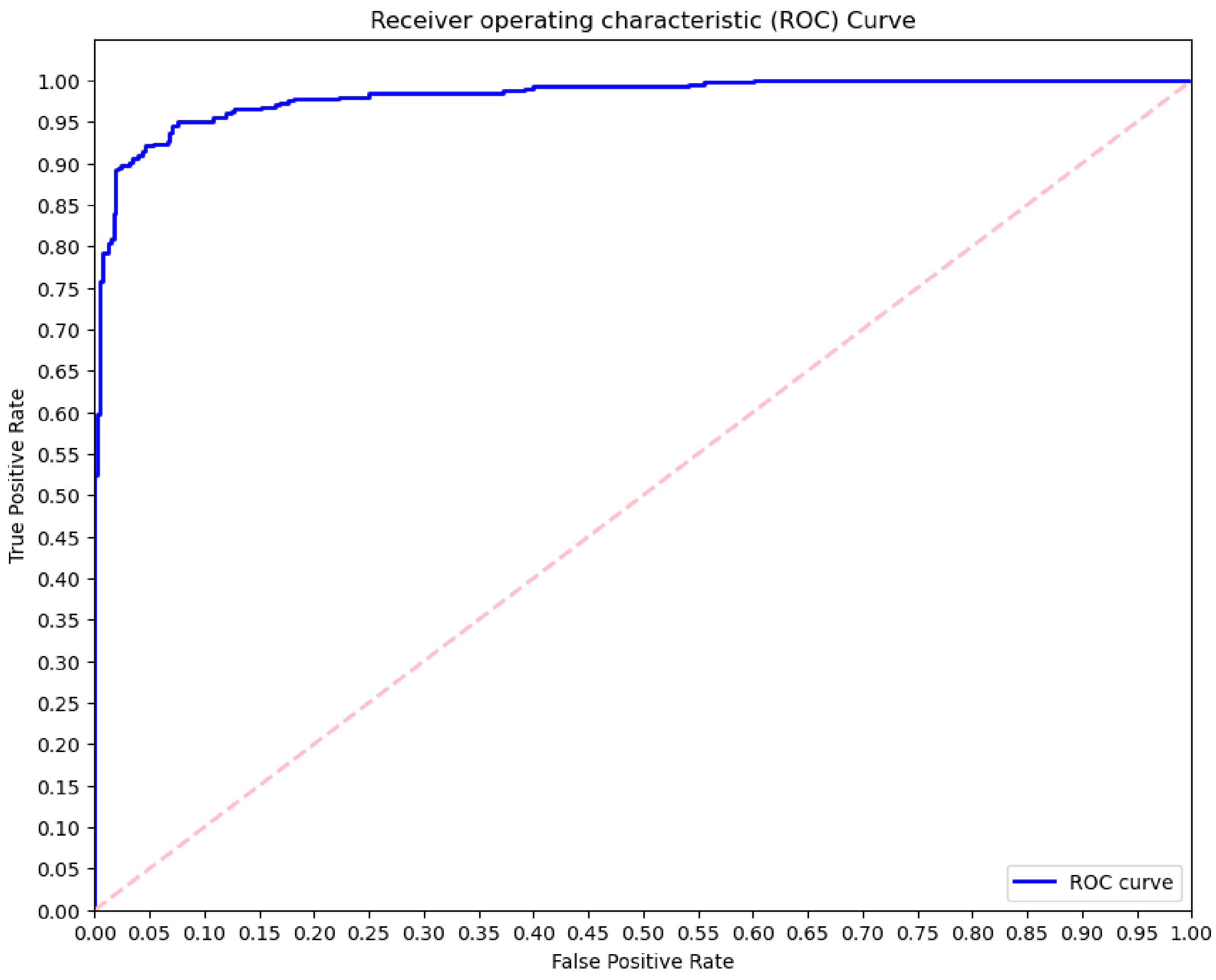
```
<Figure size 640x480 with 0 Axes>
```

Transfer Learning Predictions Check

```
In [51]: model3_predsval = pred_labels(model3, validation_generator)
```

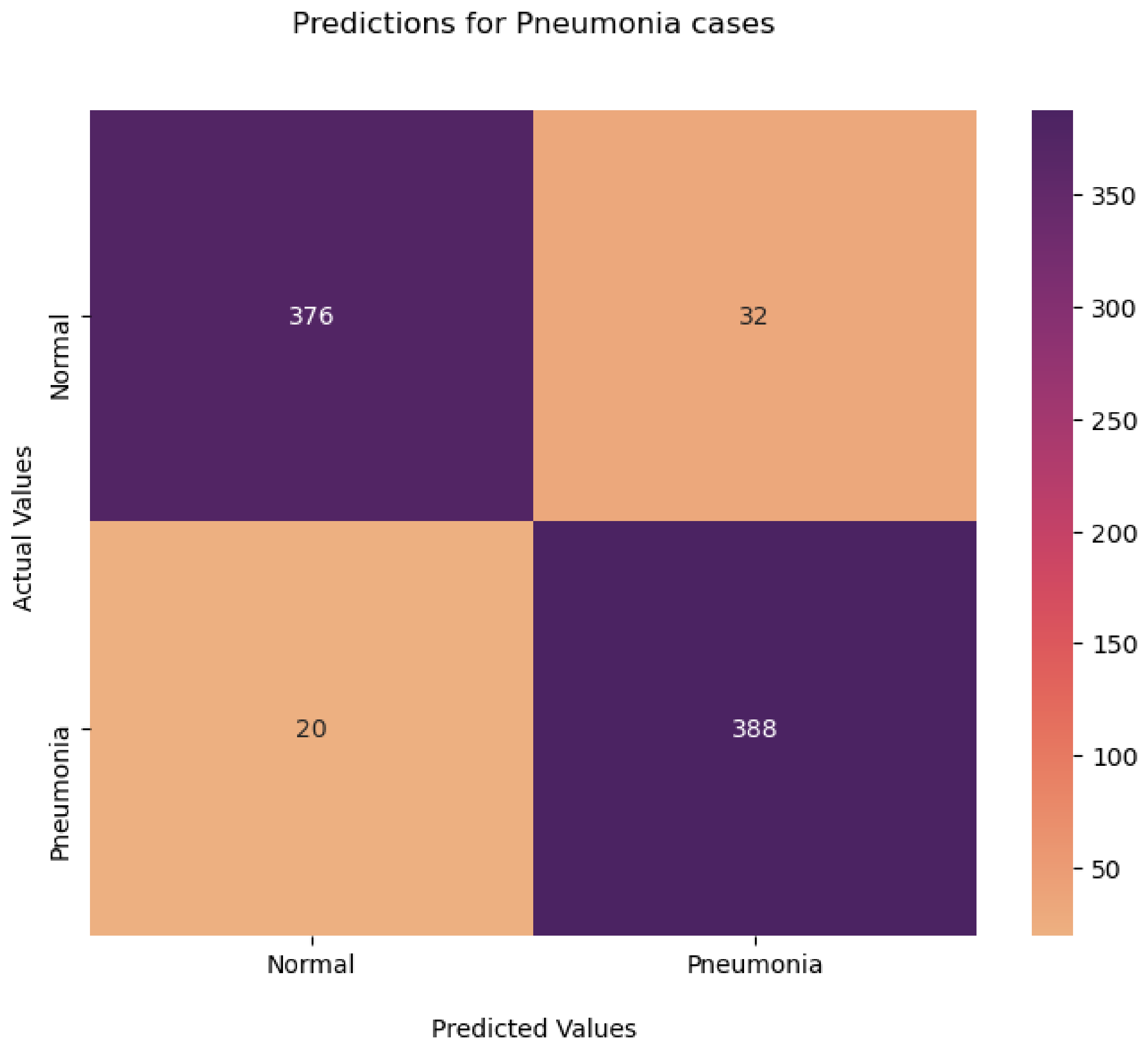
```
In [55]: plot_roc_auc(model3_predsval[0], model3_predsval[1])
```

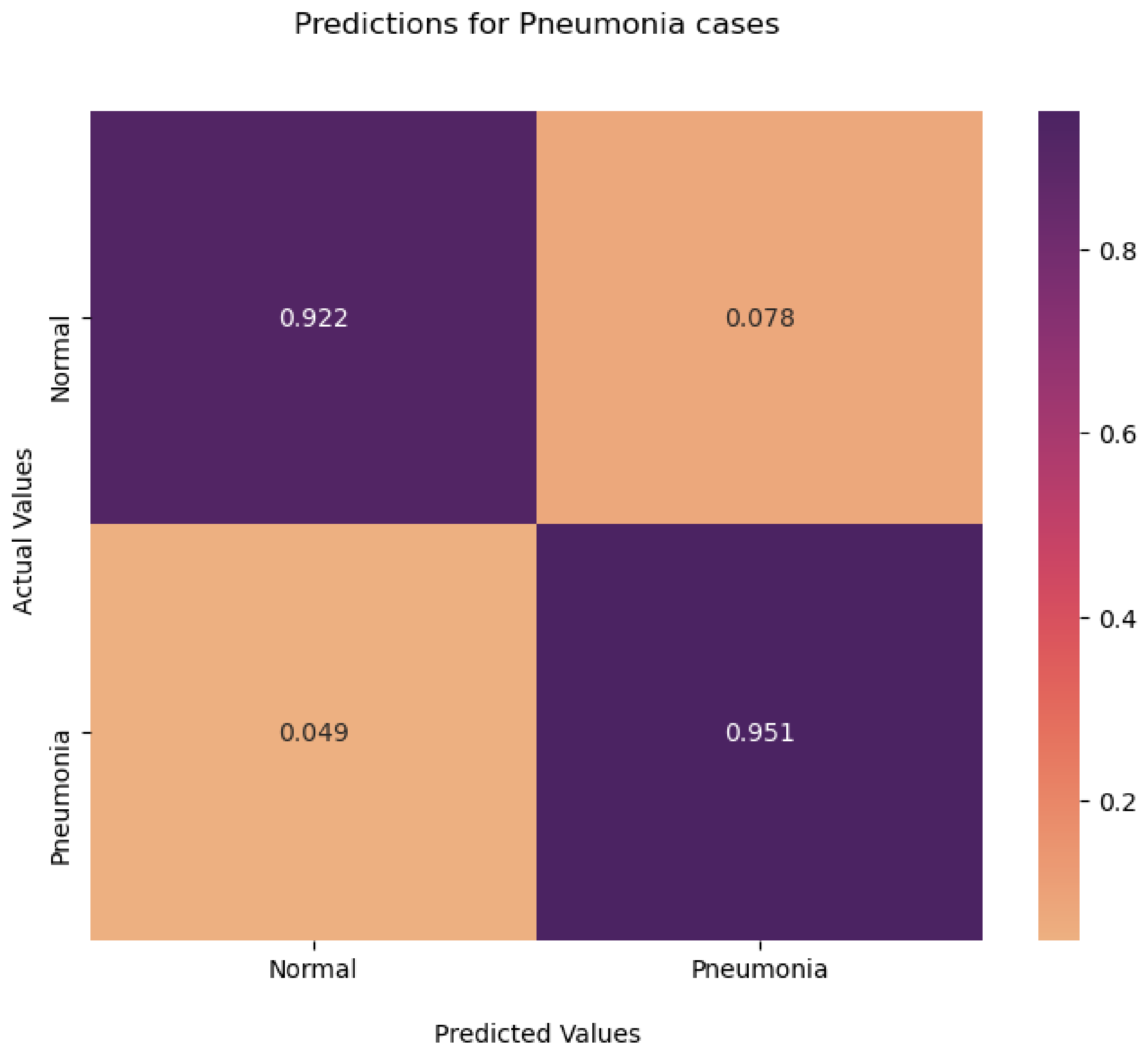
```
AUC: 0.9812872452902729
```



This graph of the ROC/AUC curve shows a slightly different ROC curve than the previous models. The elbow doesn't occur quite as blatantly or as soon, but it does climb to a higher point than the previous models.

In [88]:
```
conf_matrix(model3_predsval[0], model3_predsval[1])
```





This confusion matrix shows better results than the second model, but still doesn't have as few false negatives as the baseline model. However, there is still a 95.1% true positive rate. Now, I'm going to tune the transfer learning model, just like I did with HParams.

Transfer Learning Tuned

In [58]:
```
# Instantiating model 4

model4 = models.Sequential()

inception_v3_tuned = tf.keras.applications.InceptionV3(
    include_top=False,
    weights="imagenet",
    input_shape=(224, 224, 3),
    classes=1
)


for layer in inception_v3_tuned.layers[:-31]:
    layer.trainable = False

for i, layer in enumerate(inception_v3_tuned.layers):
    print(i, layer.name, layer.trainable)
```

```
232 conv2d_193 False
233 batch_normalization_167 False
234 activation_167 False
235 conv2d_190 False
236 conv2d_194 False
237 batch_normalization_164 False
238 batch_normalization_168 False
239 activation_164 False
240 activation_168 False
241 conv2d_191 False
242 conv2d_195 False
243 batch_normalization_165 False
244 batch_normalization_169 False
245 activation_165 False
246 activation_169 False
247 max_pooling2d_33 False
248 mixed8 False
249 conv2d_200 False
250 batch_normalization_174 False
251 activation_174 False
```

In [59]:
```
# Input Layer

model4.add(inception_v3_tuned)

# Hidden Layer
model4.add(layers.GlobalAveragePooling2D())
model4.add(layers.Dense(64, activation = 'relu'))
model4.add(layers.Dropout(0.2))

#Adding output layer
model4.add(layers.Dense(1, activation = 'sigmoid'))
```

In [60]:
```
# Compiling model
model4.compile(loss= 'binary_crossentropy',
optimizer= optimizers.Adam(lr = 1e-4),
metrics= tf.keras.metrics.BinaryAccuracy(name="binary_accuracy", dtype=None
```

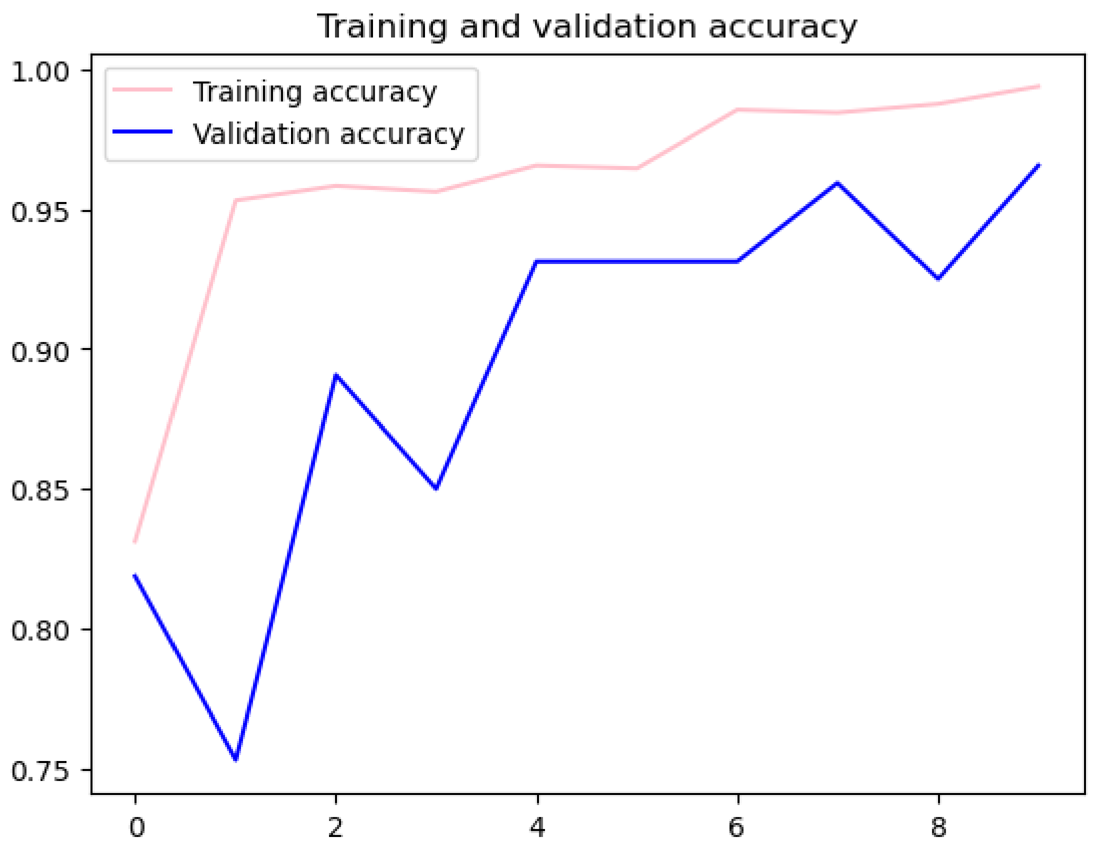
In [61]:
```
# Printing the summary for model 4

model4.summary()
```

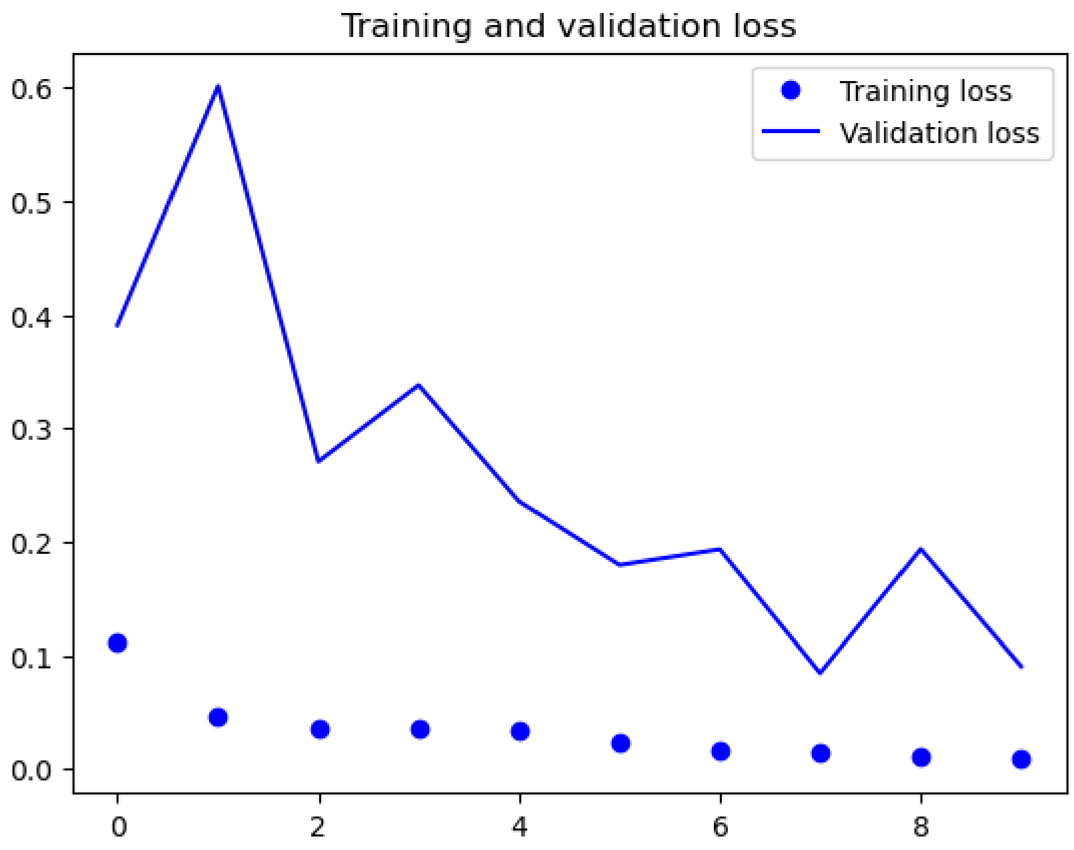
```
Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 5, 5, 2048)        21802784

_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0

_________________________________________________________________
dense_54 (Dense)             (None, 64)                131136

_________________________________________________________________
dropout_51 (Dropout)         (None, 64)                0

_________________________________________________________________
dense_55 (Dense)             (None, 1)                 65
=================================================================
Total params: 21,933,985
Trainable params: 6,204,737
Non-trainable params: 15,729,248
_________________________________________________________________
```

In [62]:
```
#Fitting model

history_transfertune=model4.fit(
train_generator, #Using train data
steps_per_epoch=30, #Keeping 30 steps
epochs=10, #Keeping 10 epochs
validation_data=validation_generator, #Using validation data
class_weight = class_weight, #Adding weights to deal with imbalance
validation_steps=10, #Keeping 10 steps
)
```

```
Epoch 1/10
30/30 [==============================] - 39s 1s/step - loss: 0.1116 - bin
ary_accuracy: 0.8313 - val_loss: 0.3909 - val_binary_accuracy: 0.8188
Epoch 2/10
30/30 [==============================] - 38s 1s/step - loss: 0.0456 - bin
ary_accuracy: 0.9531 - val_loss: 0.6015 - val_binary_accuracy: 0.7531
Epoch 3/10
30/30 [==============================] - 39s 1s/step - loss: 0.0357 - bin
ary_accuracy: 0.9583 - val_loss: 0.2706 - val_binary_accuracy: 0.8906
Epoch 4/10
30/30 [==============================] - 38s 1s/step - loss: 0.0362 - bin
ary_accuracy: 0.9563 - val_loss: 0.3380 - val_binary_accuracy: 0.8500
Epoch 5/10
30/30 [==============================] - 39s 1s/step - loss: 0.0341 - bin
ary_accuracy: 0.9656 - val_loss: 0.2355 - val_binary_accuracy: 0.9312
Epoch 6/10
30/30 [==============================] - 39s 1s/step - loss: 0.0228 - bin
ary_accuracy: 0.9646 - val_loss: 0.1793 - val_binary_accuracy: 0.9312
Epoch 7/10
30/30 [==============================] - 38s 1s/step - loss: 0.0163 - bin
ary_accuracy: 0.9854 - val_loss: 0.1932 - val_binary_accuracy: 0.9312
Epoch 8/10
30/30 [==============================] - 39s 1s/step - loss: 0.0141 - bin
ary_accuracy: 0.9844 - val_loss: 0.0842 - val_binary_accuracy: 0.9594
Epoch 9/10
30/30 [==============================] - 38s 1s/step - loss: 0.0104 - bin
ary_accuracy: 0.9875 - val_loss: 0.1934 - val_binary_accuracy: 0.9250
Epoch 10/10
30/30 [==============================] - 40s 1s/step - loss: 0.0083 - bin
ary_accuracy: 0.9937 - val_loss: 0.0902 - val_binary_accuracy: 0.9656
```

In [63]:
```
# Plotting history for model one

plot_history(history_transfertune)
```





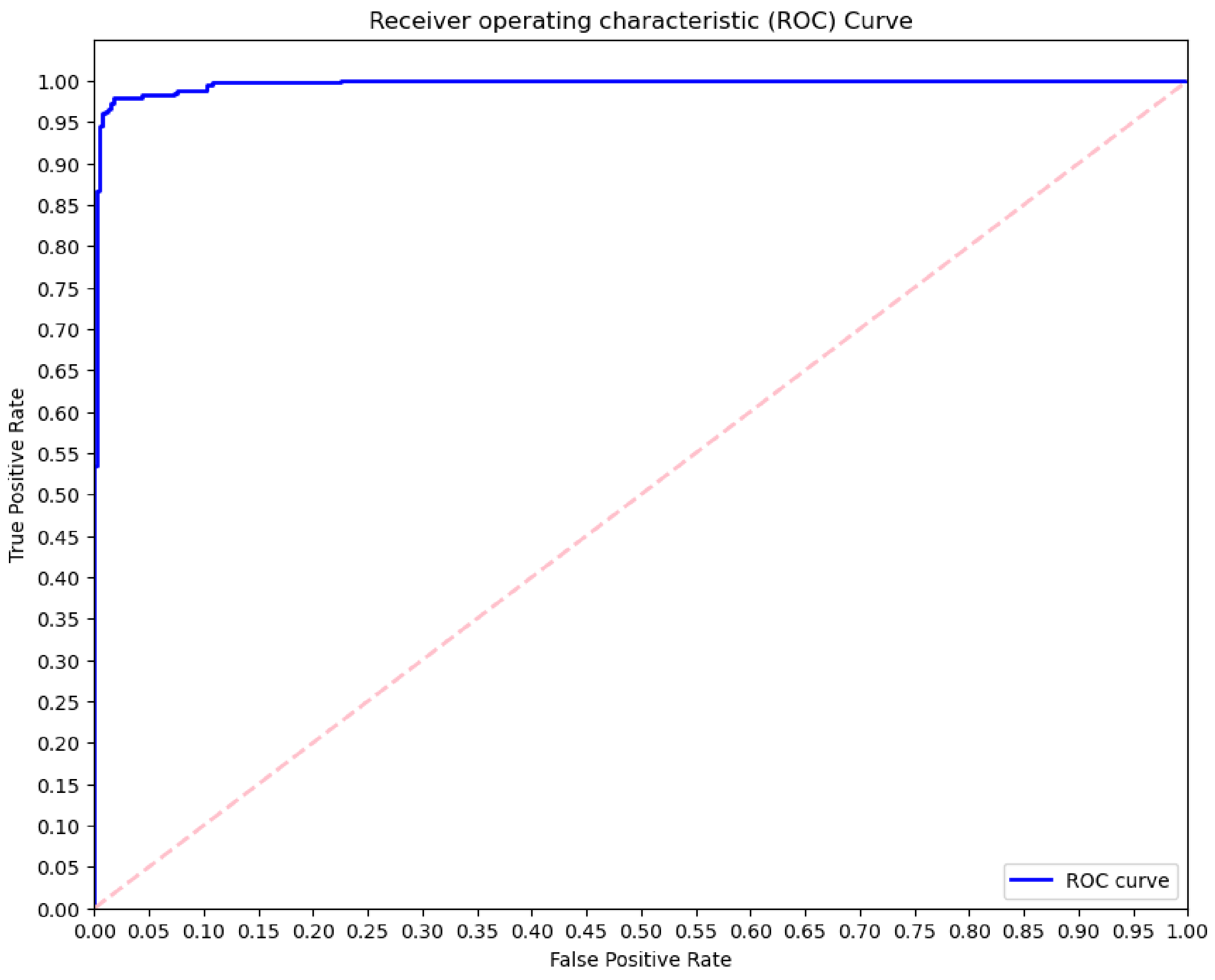
```
<Figure size 640x480 with 0 Axes>
```

Transfer Learning Predictions Check

In [92]:
```
# Calling pred_labels on model 4

model4_predsval = pred_labels(model4, validation_generator)
```

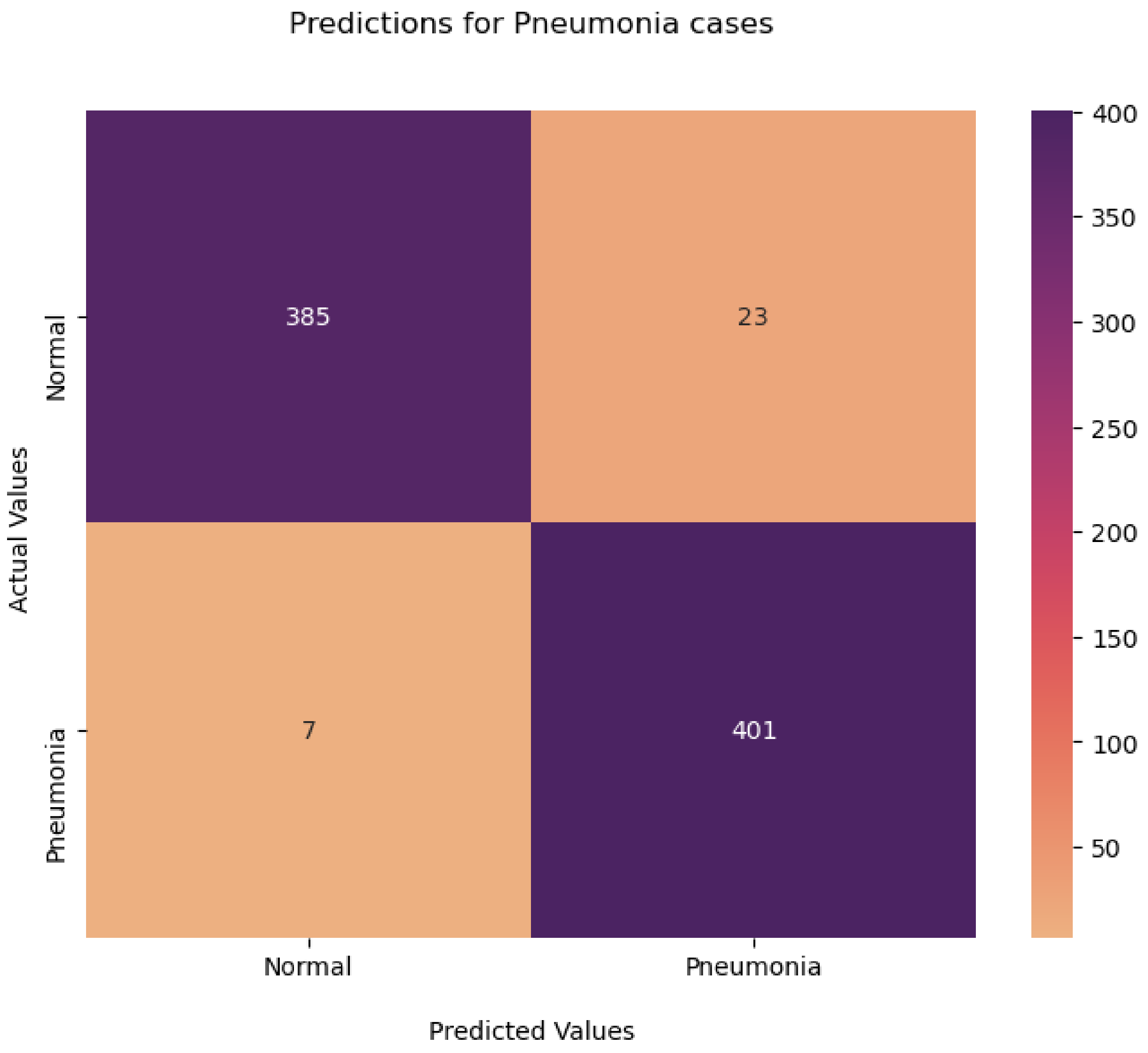
In [93]:
```
# Plotting the rocauc curbes with a function

plot_roc_auc(model4_predsval[0], model4_predsval[1])
```

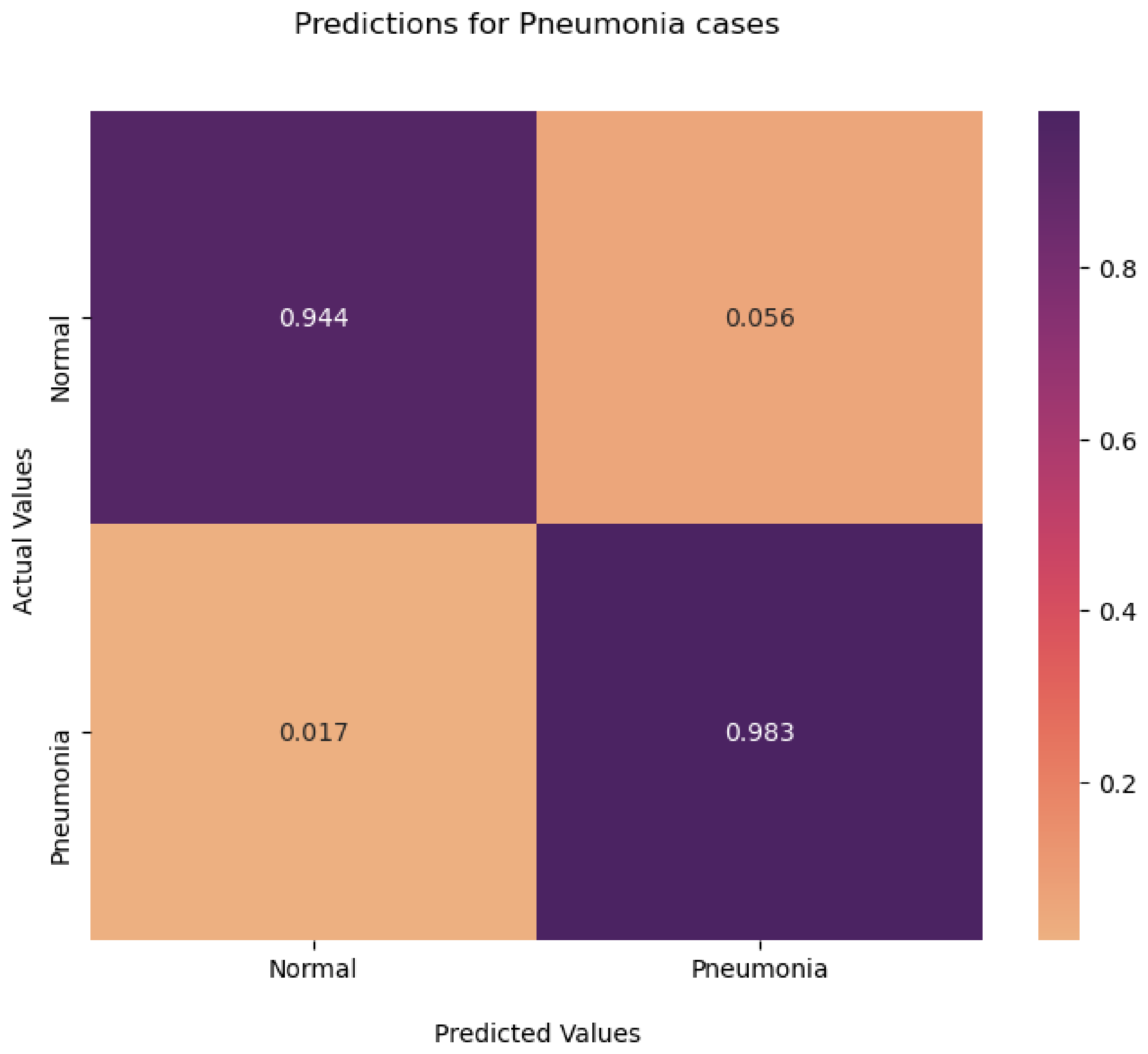
```
AUC: 0.9963535659361784
```



This ROC/AUC curve shows one of the best ROC curves that I've achieved so far, with a 99.6% AUC.

In [94]:
```
# Calling the confusion matrix function

conf_matrix(model4_predsval[0], model4_predsval[1])
```





The false negative rate in this confusion matrix for the tuned transfer learning model is 1.7%. The false positive rate is 5.6%, so still higher than the false negatives (which is good). The true positive rate is 98.3%.

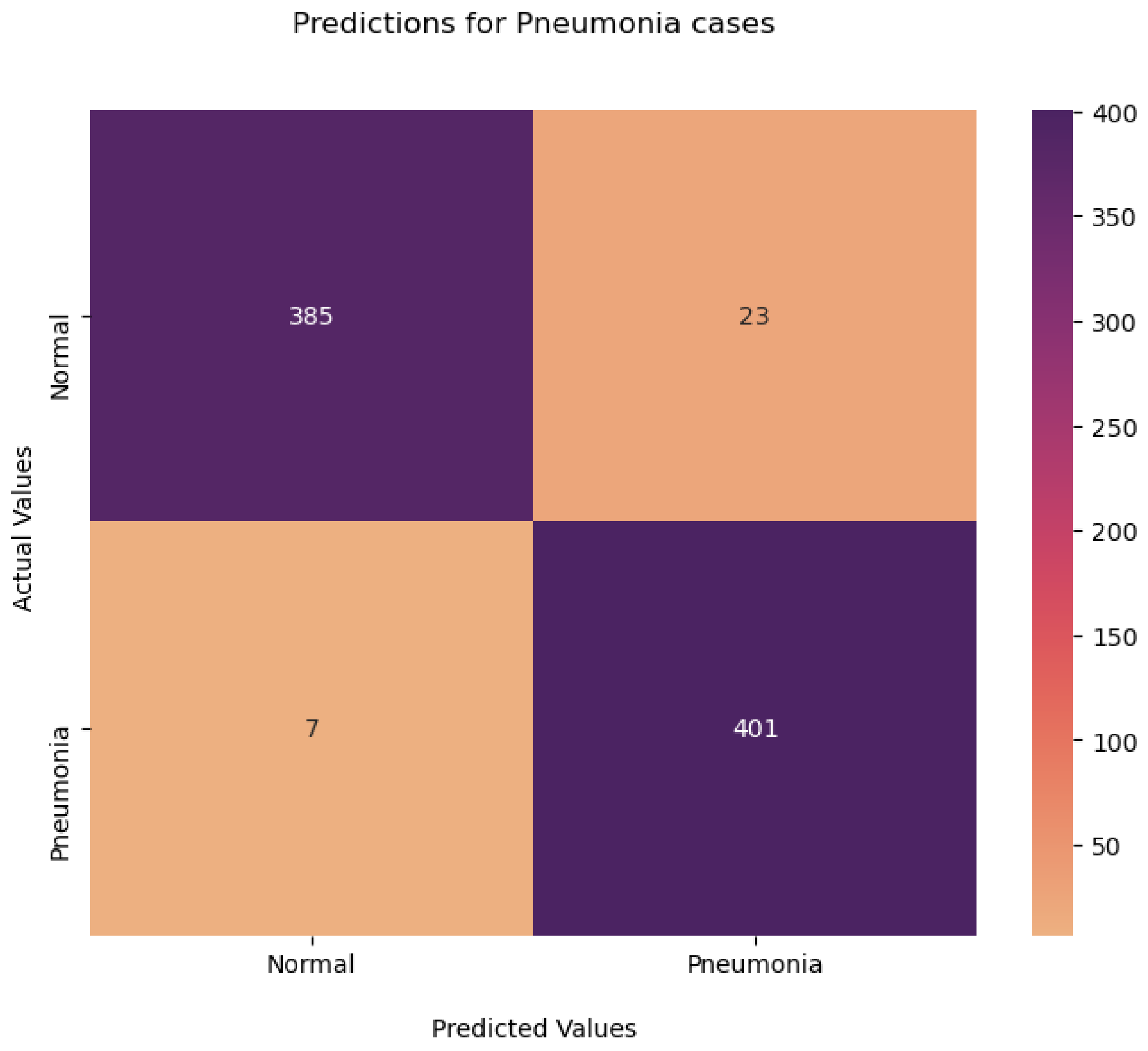
Test generator

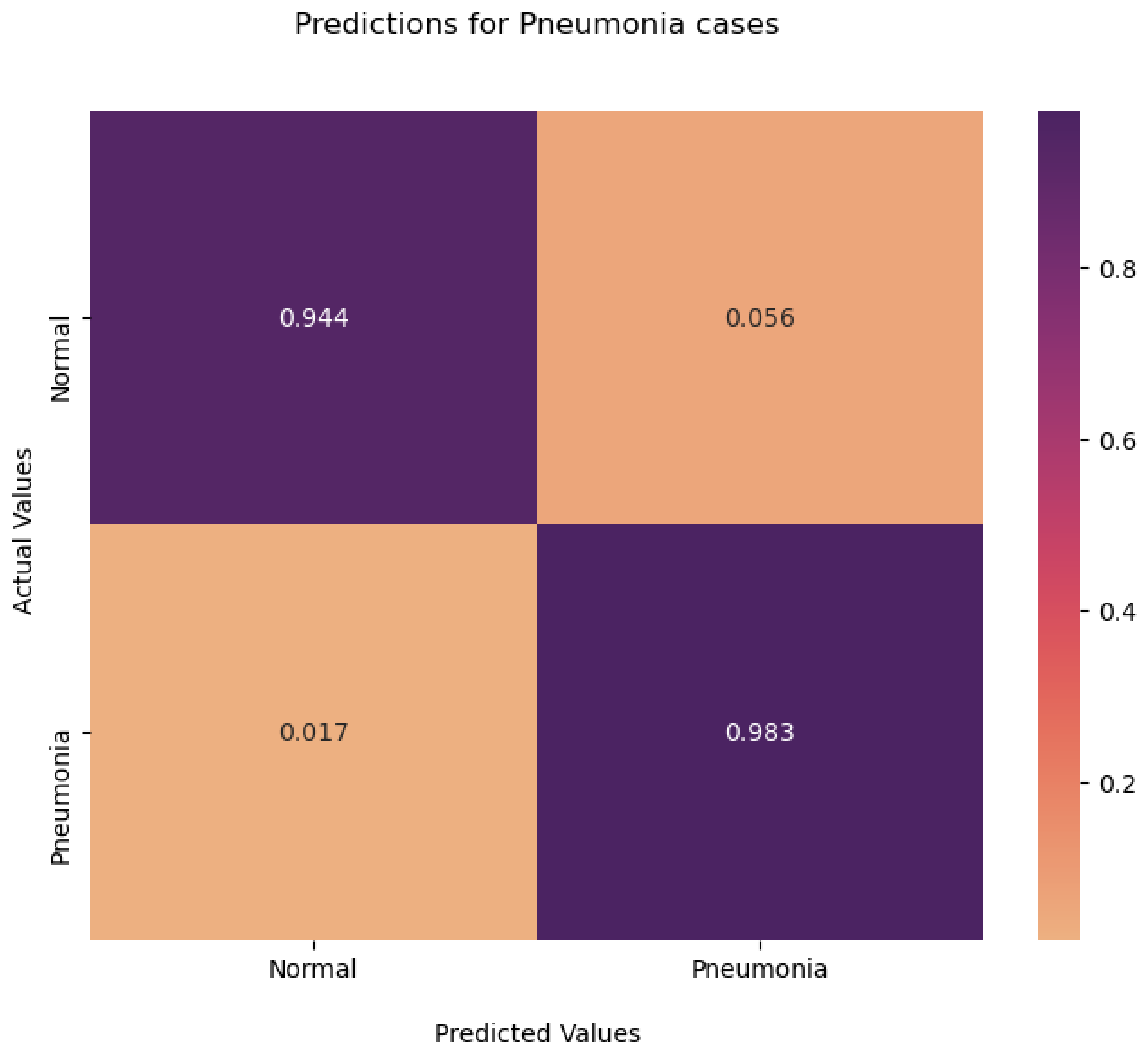
In [68]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(test_dir,
                                                        target_size=(224, 2
                                                        batch_size=32,
                                                        class_mode='binary'
                                                        shuffle = True)
```

```
Found 624 images belonging to 2 classes.
```

In [95]:
```
final_preds = pred_labels(model4, test_generator)
```

In [96]:
```
conf_matrix(model4_predsval[0], model4_predsval[1])
```





Conclusion

After iterating through different types of possible models, the final model that I've decided performs the best has a 98.3% true popsitive rate. There is a 1.7% false negative rate. This is something that I would like to improve, but I am still happy with endorsing as using it for a second opinion, or a tool to help flag pneumonia and diagnose patients faster. The AUC line had a score of 0.99635. The AUC line measures the ability of the model to distinguish between classes, which is something I am satisfied with. The last epoch of the fourth model has a binary accuracy score of 0.9937, with a validation binary accuracy score of 0.9656. Although the binary accuracy score and the validation binary accuracy score could be closer together, that is still a good performance on the validation set.

Some recommendatios I would make based on this notebook would be to **use Inception V3** as a model. In addition, there are actually several types of pneumonia. This notebook is only testing for whether or not pneumonia is present, but with more layers, different types of pneumonia could be classified as well. The reason neural networks were used for this, is because images are 3-

dimensional, so they require models that are more complex. Since there isn't patient data, I'm not able to measure how diverse the demographics are, but I would recommend those be included in