

Set Up

We'll be using the Kaggle dataset API to download our images and Google Drive to store them, so the first several lines are just a bit of maintenance and prep work do that. We performed our work in Google Colab, as it offers hosted GPUs that make CNN training much, much faster, and some of that prep work is also contained in the first few lines.

```
In [ ]: ! pip install -q kaggle
```

```
In [ ]: from google.colab import files
files.upload()
```

No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving kaggle.json to kaggle.json

```
Out[ ]: {'kaggle.json': b'{"username":"mcn9284","key":"86a8a5f625a901c2d5e685a9c20c777f"}'}
```

```
In [ ]: ! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
```

```
In [ ]: ! chmod 600 ~/.kaggle/kaggle.json
```

```
In [ ]: ! kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
```

Downloading chest-xray-pneumonia.zip to /content
99% 2.27G/2.29G [00:14<00:00, 224MB/s]
100% 2.29G/2.29G [00:14<00:00, 167MB/s]

```
In [ ]: ! unzip chest-xray-pneumonia.zip && rm chest-xray-pneumonia.zip
```

Streaming output truncated to the last 5000 lines.

```
inflating: chest_xray/train/NORMAL/IM-0435-0001-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0435-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0437-0001-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0437-0001-0002.jpeg
inflating: chest_xray/train/NORMAL/IM-0437-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0438-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0439-0001-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0439-0001-0002.jpeg
inflating: chest_xray/train/NORMAL/IM-0439-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0440-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0441-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0442-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0444-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0445-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0446-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0447-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0448-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0449-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0450-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0451-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0452-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0453-0001-0002.jpeg
inflating: chest_xray/train/NORMAL/IM-0453-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0455-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0456-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0457-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0458-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0459-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0460-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0461-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0463-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0464-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0465-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0466-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0467-0001-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0467-0001-0002.jpeg
inflating: chest_xray/train/NORMAL/IM-0467-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0469-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0471-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0472-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0473-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0474-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0475-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0476-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0477-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0478-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0479-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0480-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0481-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0482-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0483-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0484-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0485-0001.jpeg
inflating: chest_xray/train/NORMAL/IM-0486-0001.jpeg
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

In []:

```
from IPython.core.interactiveshell import InteractiveShell
```

```

import os
import cv2
import glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from tensorflow.keras import models, layers, optimizers, regularizers, activations
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, RandomContrast, RandomBrightness, AveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from google.colab import drive
import warnings
warnings.filterwarnings("ignore")

InteractiveShell.ast_node_interactivity = 'all'

```

```

In [ ]:
# Mounting Google Drive
drive.mount('gdrive', force_remount=True)

```

```

-----
NameError                                Traceback (most recent call last)
<ipython-input-10-91b04f9badff> in <module>
      1 # Mounting Google Drive
----> 2 drive.mount('gdrive', force_remount=True)

NameError: name 'drive' is not defined

```

Data Understanding

Our data is stored in directory "chest_x_ray" and is further subdivided into three subdirectories--train, val, and test. Each of those is further divided into 'NORMAL' and 'PNEUMONIA' folders. The data itself consists of jpeg images of X-Rays from both patients with and without pneumonia.

Generally speaking, pneumonia presents on radiograph as areas of diffuse opacity (sometimes called 'ground glass opacity') which represents areas of fluid accumulation in and around the alveoli. This fluid is created by the immune system in an attempt to rid the body of the bacterial or viral invader, and has the unfortunate side-effect of lowering the lungs' ability to exchange oxygen and carbon dioxide. Needless to say, not a good thing.

Our task is to create a convoluted neural network that can read these radiographs and classify them into either 'normal' or 'pneumonia' categories.

Preprocessing

We will first create paths to each folder of images, and create our data using generators from those directories. We then associate the labels and images and collapse them into two variables (X for images, y for labels) so we can create a better proportioned train-test-split.

```

In [ ]:
# Path to data
from pathlib import Path
data_dir = Path('chest_xray/')

# Path to train directory
train_dir = data_dir / 'train'

# Path to validation directory
val_dir = data_dir / 'val'

# Path to test directory
test_dir = data_dir / 'test'

```

```

In [ ]:
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    train_dir,
    batch_size=5216)

test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_dir,
    batch_size=624)

val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    val_dir,
    batch_size=16)

```

```

Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 16 images belonging to 2 classes.

```

```

In [ ]:
train_img, train_labels = next(train_generator)
test_img, test_labels = next(test_generator)

```

```
val_img, val_labels = next(val_generator)
```

```
In [ ]: tr_y = np.reshape(train_labels[:,0], (5216,1))
        te_y = np.reshape(test_labels[:,0], (624,1))
        val_y = np.reshape(val_labels[:,0], (16,1))
```

```
In [ ]: X = np.concatenate((train_img, test_img, val_img))
        y = np.concatenate((tr_y, te_y, val_y))
```

Let's get a sense of what we're working with. First we will visualize our class balance, then we will take a look at several images from each class.

Since we've got a bit of an imbalance here, we will work out the class weights, assign them appropriately, and include them in our parameters later on when we configure our neural networks.

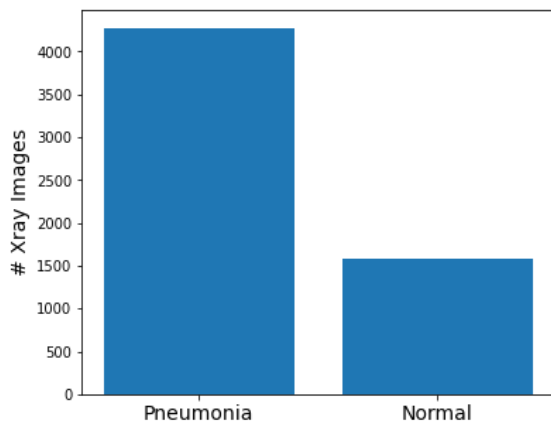
```
In [ ]: counts = np.unique(y, return_counts=True)[1]
        fig, ax = plt.subplots(figsize=(6, 5))
        plt.bar(['Pneumonia', 'Normal'], counts)
        plt.ylabel('# Xray Images', fontsize=14)
        plt.xticks(fontsize=14)
        print("Pneumonia: ", counts[0])
        print("Normal: ", counts[1])
```

```
Out[ ]: <BarContainer object of 2 artists>
```

```
Out[ ]: Text(0, 0.5, '# Xray Images')
```

```
Out[ ]: ([0, 1], <a list of 2 Text major ticklabel objects>)
```

```
Pneumonia: 4273
Normal: 1583
```



```
In [ ]: def class_weights(n, nj):
        return n / (2 * nj)
```

```
In [ ]: class_weights(5856, 4273)
        print()
        class_weights(5856, 1583)
```

```
Out[ ]: 0.6852328574771823
```

```
Out[ ]: 1.8496525584333543
```

```
In [ ]: pneumonia_weights = print(
        'weight class 1: ', round(class_weights(5856, 4273), 2))
        normal_weights = print('weight class 0: ', round(class_weights(5856, 1583), 2))
```

```
weight class 1: 0.69
weight class 0: 1.85
```

```
In [ ]: class_weight = {0:0.69, 1:1.85}
```

Now lets have a look at some of the xrays.

First we'll put the images and their associated labels into a Dataframe, and from there select five randomly from each class

```
In [ ]: normal_cases_dir = train_dir / 'NORMAL'
        pneumonia_cases_dir = train_dir / 'PNEUMONIA'

        # List all normal and pneumonia images
        normal_cases = normal_cases_dir.glob('*.jpeg')
        pneumonia_cases = pneumonia_cases_dir.glob('*.jpeg')
```

```

# Create empty list
train_data = []

# Label normal images 0
for img in normal_cases:
    train_data.append((img, 0))

# Label pneumonia images 1
for img in pneumonia_cases:
    train_data.append((img, 1))

# Create dataframe with image and label data
train_data = pd.DataFrame(train_data, columns=['image', 'label'], index=None)

# Shuffle the data
train_data = train_data.sample(frac=1.).reset_index(drop=True)

# Inspect dataframe
train_data.head()

```

```

Out[ ]:

```

	image	label
0	chest_xray/train/PNEUMONIA/person1441_virus_24...	1
1	chest_xray/train/PNEUMONIA/person41_bacteria_2...	1
2	chest_xray/train/NORMAL/IM-0740-0001.jpeg	0
3	chest_xray/train/NORMAL/NORMAL2-IM-0966-0001.jpeg	0
4	chest_xray/train/PNEUMONIA/person494_bacteria_...	1

```

In [ ]:
from skimage.io import imread

# Create lists of the first five rows in each class
pneumonia_samples = (
    train_data[train_data['label'] == 1]['image'].iloc[:5]).tolist()
normal_samples = (train_data[train_data['label']
    == 0]['image'].iloc[:5]).tolist()

# Concatenate and delete the two lists
samples = pneumonia_samples + normal_samples
del pneumonia_samples, normal_samples

# Plot data
f, ax = plt.subplots(2, 5, figsize=(30, 10))
for i in range(10):
    img = imread(samples[i])
    ax[i//5, i % 5].imshow(img, cmap='gray')
    if i < 5:
        ax[i//5, i % 5].set_title("Pneumonia")
    else:
        ax[i//5, i % 5].set_title("Normal")
    ax[i//5, i % 5].axis('off')
    ax[i//5, i % 5].set_aspect('auto')
plt.show()

```

```

Out[ ]: <matplotlib.image.AxesImage at 0x7efc72fe6c70>

```

```

Out[ ]: Text(0.5, 1.0, 'Pneumonia')

```

```

Out[ ]: (-0.5, 1759.5, 1231.5, -0.5)

```

```

Out[ ]: <matplotlib.image.AxesImage at 0x7efc734bfa90>

```

```

Out[ ]: Text(0.5, 1.0, 'Pneumonia')

```

```

Out[ ]: (-0.5, 1071.5, 679.5, -0.5)

```

```

Out[ ]: <matplotlib.image.AxesImage at 0x7efc7346de50>

```

```

Out[ ]: Text(0.5, 1.0, 'Pneumonia')

```

```

Out[ ]: (-0.5, 1579.5, 1492.5, -0.5)

```

```

Out[ ]: <matplotlib.image.AxesImage at 0x7efc7312f280>

```

```

Out[ ]: Text(0.5, 1.0, 'Pneumonia')

```

```

Out[ ]: (-0.5, 815.5, 567.5, -0.5)

```

```

Out[ ]: <matplotlib.image.AxesImage at 0x7efc730dd610>

```

```

Out[ ]: Text(0.5, 1.0, 'Pneumonia')

```

```

Out[ ]: (-0.5, 1395.5, 1098.5, -0.5)

```

```

Out[ ]: <matplotlib.image.AxesImage at 0x7efc7310ca00>

```

```

Out[ ]: Text(0.5, 1.0, 'Normal')

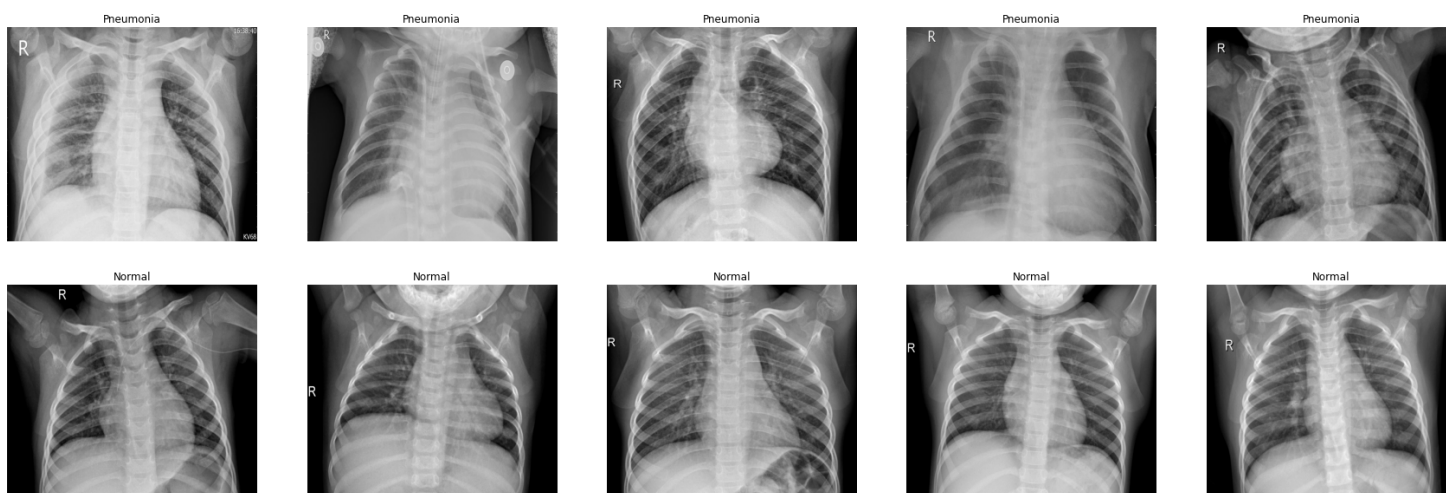
```

```

Out[ ]: (-0.5, 1839.5, 1282.5, -0.5)

```

```
Out[ ]: <matplotlib.image.AxesImage at 0x7efc730bbdf0>
Out[ ]: Text(0.5, 1.0, 'Normal')
Out[ ]: (-0.5, 1517.5, 1070.5, -0.5)
Out[ ]: <matplotlib.image.AxesImage at 0x7efc730741f0>
Out[ ]: Text(0.5, 1.0, 'Normal')
Out[ ]: (-0.5, 1769.5, 1471.5, -0.5)
Out[ ]: <matplotlib.image.AxesImage at 0x7efc730235e0>
Out[ ]: Text(0.5, 1.0, 'Normal')
Out[ ]: (-0.5, 1939.5, 1502.5, -0.5)
Out[ ]: <matplotlib.image.AxesImage at 0x7efc735b9a60>
Out[ ]: Text(0.5, 1.0, 'Normal')
Out[ ]: (-0.5, 1927.5, 1302.5, -0.5)
```



There are definite differences between the two classes visually, with pneumonia xrays showing the characteristic 'ground glass opacity'. There are a few cases that, at least to the untrained eye, appear to be ambiguous. Hopefully our model can pick up on features we can't.

The last step of our preprocessing process is to split our data. Recall just a bit ago that we collapsed all the images and labels into X and y variables without respect to test or validation datasets. The idea was that, since the data had been apportioned so awfully, we'd bypass that dumpster fire and split the data in a way that's easier to work with.

We use a `train_test_split`, set the `test_size` to 20% of the total data, and shuffle!

Then of course we check the shape of our splits.

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.20, shuffle=True, random_state=42)
```

```
In [ ]: print('Training Shape: ', X_train.shape)
        print("Test Shape: ", X_test.shape)
```

```
Training Shape: (4684, 256, 256, 3)
Test Shape: (1172, 256, 256, 3)
```

```
In [ ]: keras.backend.clear_session()
```

Modeling

Our baseline model will start with one convolution/pooling layer, flattened into a 32 unit fully connected layer, which feeds into an output layer with sigmoid activation.

Our conv layer has a filter size of 5x5 because we want the network to have a larger receptive field and to pick up on more general features (at least to start!).

We use an adam optimizer with learning rate set at 0.0001.

Model 1

```
In [ ]: callbacks = EarlyStopping(monitor='val_loss', min_delta=0.05, mode='min',patience=10)
```

```
In [ ]: cnn = keras.Sequential()
```

```

cnn.add(layers.Conv2D(8, (5, 5), activation='relu', input_shape=(256, 256, 3)))
cnn.add(layers.MaxPooling2D(2, 2))

cnn.add(layers.Flatten())
cnn.add(layers.Dense(32, activation='relu'))
cnn.add(layers.Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(learning_rate=1e-4)
cnn.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])

```

```

In [ ]:
cnn.build(input_shape=(None, 256, 256, 3))
cnn.summary()

```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 252, 252, 8)	608
max_pooling2d_1 (MaxPooling2D)	(None, 126, 126, 8)	0
flatten_1 (Flatten)	(None, 127008)	0
dense_2 (Dense)	(None, 32)	4064288
dense_3 (Dense)	(None, 1)	33

=====
 Total params: 4,064,929
 Trainable params: 4,064,929
 Non-trainable params: 0

```

In [ ]:
history = cnn.fit(X_train,
                  y_train,
                  epochs=100,
                  batch_size=64,
                  class_weight=class_weight,
                  validation_split=0.20,
                  callbacks=callbacks,
                  verbose=1)

```

Epoch 1/100
 59/59 [=====] - 2s 33ms/step - loss: 0.4871 - accuracy: 0.7785 - val_loss: 0.4975 - val_accuracy: 0.7855
 Epoch 2/100
 59/59 [=====] - 1s 23ms/step - loss: 0.2633 - accuracy: 0.9026 - val_loss: 0.2481 - val_accuracy: 0.9168
 Epoch 3/100
 59/59 [=====] - 1s 23ms/step - loss: 0.1934 - accuracy: 0.9295 - val_loss: 0.2467 - val_accuracy: 0.9136
 Epoch 4/100
 59/59 [=====] - 1s 24ms/step - loss: 0.1515 - accuracy: 0.9464 - val_loss: 0.1704 - val_accuracy: 0.9381
 Epoch 5/100
 59/59 [=====] - 1s 23ms/step - loss: 0.1394 - accuracy: 0.9525 - val_loss: 0.2224 - val_accuracy: 0.9210
 Epoch 6/100
 59/59 [=====] - 1s 23ms/step - loss: 0.1225 - accuracy: 0.9573 - val_loss: 0.1582 - val_accuracy: 0.9434
 Epoch 7/100
 59/59 [=====] - 1s 23ms/step - loss: 0.1171 - accuracy: 0.9568 - val_loss: 0.1369 - val_accuracy: 0.9552
 Epoch 8/100
 59/59 [=====] - 1s 24ms/step - loss: 0.1078 - accuracy: 0.9637 - val_loss: 0.1489 - val_accuracy: 0.9456
 Epoch 9/100
 59/59 [=====] - 1s 23ms/step - loss: 0.1041 - accuracy: 0.9634 - val_loss: 0.1303 - val_accuracy: 0.9562
 Epoch 10/100
 59/59 [=====] - 1s 24ms/step - loss: 0.1014 - accuracy: 0.9632 - val_loss: 0.1285 - val_accuracy: 0.9498
 Epoch 11/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0855 - accuracy: 0.9693 - val_loss: 0.1217 - val_accuracy: 0.9573
 Epoch 12/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0826 - accuracy: 0.9696 - val_loss: 0.1287 - val_accuracy: 0.9498
 Epoch 13/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0788 - accuracy: 0.9738 - val_loss: 0.1308 - val_accuracy: 0.9552
 Epoch 14/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0825 - accuracy: 0.9709 - val_loss: 0.1160 - val_accuracy: 0.9626
 Epoch 15/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0732 - accuracy: 0.9744 - val_loss: 0.1208 - val_accuracy: 0.9541
 Epoch 16/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0741 - accuracy: 0.9733 - val_loss: 0.1323 - val_accuracy: 0.9488
 Epoch 17/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0644 - accuracy: 0.9800 - val_loss: 0.1290 - val_accuracy: 0.9509
 Epoch 18/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0616 - accuracy: 0.9811 - val_loss: 0.1358 - val_accuracy: 0.9509
 Epoch 19/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0541 - accuracy: 0.9837 - val_loss: 0.1277 - val_accuracy: 0.9488
 Epoch 20/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0578 - accuracy: 0.9829 - val_loss: 0.1435 - val_accuracy: 0.9488
 Epoch 21/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0493 - accuracy: 0.9875 - val_loss: 0.1128 - val_accuracy: 0.9637
 Epoch 22/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0442 - accuracy: 0.9885 - val_loss: 0.1177 - val_accuracy: 0.9541
 Epoch 23/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0431 - accuracy: 0.9885 - val_loss: 0.1308 - val_accuracy: 0.9530
 Epoch 24/100
 59/59 [=====] - 1s 23ms/step - loss: 0.0400 - accuracy: 0.9912 - val_loss: 0.1119 - val_accuracy: 0.9658

The following two functions allow us to evaluate our models as we progress.

The first plots our train and validation loss and accuracies over epoch.

The second generates a confusion matrix that provides a breakdown of predicted and actual classifications. This is where we get our true positives and false negatives etc.

```
In [ ]: def learn_curves(history):
# get history of model
history_dict = history.history

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# define accruacy and loss values
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

# set x range
epochs = range(1, len(acc_values) + 1)

# plot accuracy for training and validation
ax[0].plot(epochs, acc_values, label='Training acc')
ax[0].plot(epochs, val_acc_values, label='Validation acc')
ax[0].set_title('Training & Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend()

# plot loss for training and validation
ax[1].plot(epochs, loss_values, label='Training loss')
ax[1].plot(epochs, val_loss_values, label='Validation loss')
ax[1].set_title('Training & Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend()

print("Final Training Accuracy:", acc_values[-1])
print("Final Validation Accuracy:", val_acc_values[-1])
```

```
In [ ]: def evaluation(model):
# make predictions on X_test and convert probabilities to binary classes
predictions = (model.predict(X_test) > 0.5)*1

# generate numpy array confusion matrix
cm = confusion_matrix(y_test, predictions)

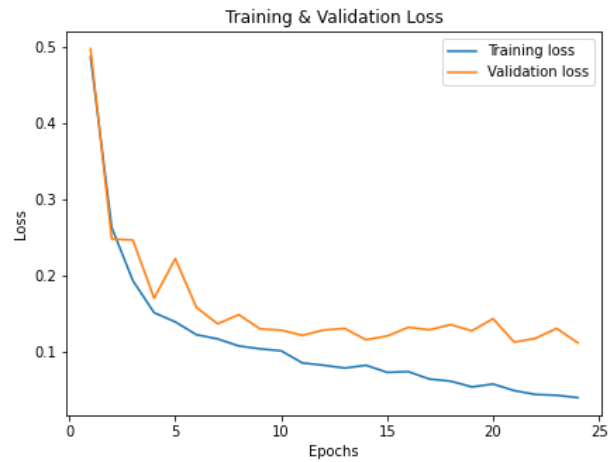
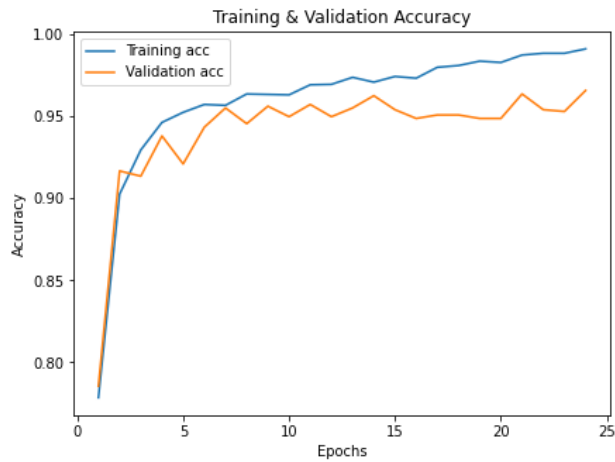
# convert array to dataframe and transpose confusion matrix
df_cm = pd.DataFrame(cm.T, index=['Pneumonia', 'Normal'],
                      columns=['Pneumonia', 'Normal'])

# plot cm as heatmap for aesthetics and readability
sns.heatmap(df_cm.T, annot=True, fmt='d', cmap='Blues',
            linecolor='black', linewidths=1)
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.ylim(0, 2)
plt.xlim(2, 0)
plt.show()

print(model.evaluate(X_test, y_test))
```

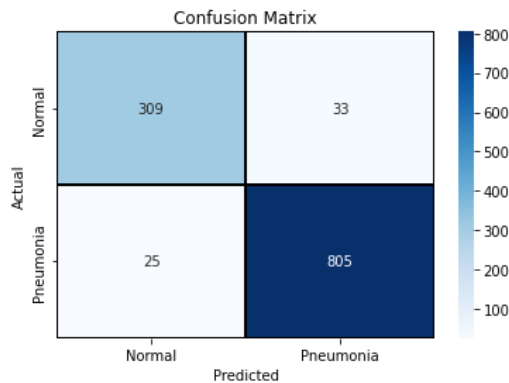
```
In [ ]: learn_curves(history)
```

```
Final Training Accuracy: 0.9911929368972778
Final Validation Accuracy: 0.965848445892334
```



```
In [ ]: evaluation(cnn)
```

37/37 [=====] - 1s 14ms/step



37/37 [=====] - 0s 9ms/step - loss: 0.1256 - accuracy: 0.9505
[0.12564462423324585, 0.9505119323730469]

So, clearly we have some overfitting. We will introduce a regularization term on the conv layer and see if we can get that under some control.

Overfitting aside, these are encouraging results!

Model 2

Our first iteration retains the basic architecture of the baseline, but introduces a regularizer in the conv2d layer. L2 regularization lessens the effect of the nodes on the outputs.

```
In [ ]: cnn2 = keras.Sequential()

cnn2.add(layers.Conv2D(8, (5, 5), activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01), input_shape=(256, 256, 3)))
cnn2.add(layers.MaxPooling2D(2, 2))

cnn2.add(layers.Flatten())
cnn2.add(layers.Dense(32, activation='relu'))
cnn2.add(layers.Dense(units=1, activation='sigmoid'))

opt = keras.optimizers.Adam(learning_rate=1e-4)
cnn2.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn2.build(input_shape=(None, 256, 256, 3))
cnn2.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 252, 252, 8)	608
max_pooling2d_3 (MaxPooling2D)	(None, 126, 126, 8)	0
flatten_3 (Flatten)	(None, 127008)	0
dense_6 (Dense)	(None, 32)	4064288
dense_7 (Dense)	(None, 1)	33

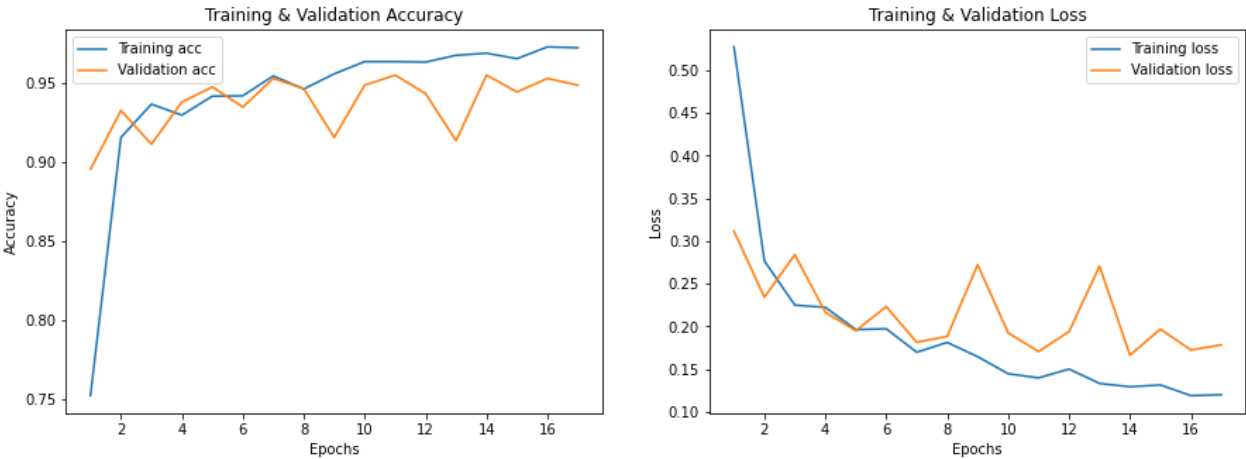
=====
Total params: 4,064,929
Trainable params: 4,064,929
Non-trainable params: 0

```
In [ ]: history2 = cnn2.fit(X_train,
                           y_train,
                           epochs=100,
                           batch_size=64,
                           class_weight=class_weight,
                           validation_split=0.20,
                           callbacks=callbacks,
                           verbose=1)
```

Epoch 1/100
59/59 [=====] - 2s 31ms/step - loss: 0.5270 - accuracy: 0.7521 - val_loss: 0.3117 - val_accuracy: 0.8954
Epoch 2/100
59/59 [=====] - 1s 23ms/step - loss: 0.2768 - accuracy: 0.9157 - val_loss: 0.2342 - val_accuracy: 0.9328
Epoch 3/100
59/59 [=====] - 1s 23ms/step - loss: 0.2251 - accuracy: 0.9367 - val_loss: 0.2841 - val_accuracy: 0.9114
Epoch 4/100
59/59 [=====] - 1s 23ms/step - loss: 0.2225 - accuracy: 0.9298 - val_loss: 0.2167 - val_accuracy: 0.9381
Epoch 5/100
59/59 [=====] - 1s 23ms/step - loss: 0.1965 - accuracy: 0.9418 - val_loss: 0.1952 - val_accuracy: 0.9477
Epoch 6/100
59/59 [=====] - 1s 23ms/step - loss: 0.1974 - accuracy: 0.9421 - val_loss: 0.2233 - val_accuracy: 0.9349
Epoch 7/100
59/59 [=====] - 1s 23ms/step - loss: 0.1700 - accuracy: 0.9546 - val_loss: 0.1817 - val_accuracy: 0.9530
Epoch 8/100
59/59 [=====] - 1s 23ms/step - loss: 0.1814 - accuracy: 0.9464 - val_loss: 0.1885 - val_accuracy: 0.9466
Epoch 9/100
59/59 [=====] - 1s 23ms/step - loss: 0.1648 - accuracy: 0.9560 - val_loss: 0.2724 - val_accuracy: 0.9157
Epoch 10/100
59/59 [=====] - 1s 23ms/step - loss: 0.1448 - accuracy: 0.9637 - val_loss: 0.1927 - val_accuracy: 0.9488
Epoch 11/100
59/59 [=====] - 1s 23ms/step - loss: 0.1400 - accuracy: 0.9637 - val_loss: 0.1708 - val_accuracy: 0.9552
Epoch 12/100
59/59 [=====] - 1s 23ms/step - loss: 0.1503 - accuracy: 0.9634 - val_loss: 0.1941 - val_accuracy: 0.9434
Epoch 13/100
59/59 [=====] - 1s 23ms/step - loss: 0.1335 - accuracy: 0.9677 - val_loss: 0.2707 - val_accuracy: 0.9136
Epoch 14/100
59/59 [=====] - 1s 23ms/step - loss: 0.1296 - accuracy: 0.9690 - val_loss: 0.1666 - val_accuracy: 0.9552
Epoch 15/100
59/59 [=====] - 1s 23ms/step - loss: 0.1317 - accuracy: 0.9656 - val_loss: 0.1970 - val_accuracy: 0.9445
Epoch 16/100
59/59 [=====] - 1s 23ms/step - loss: 0.1192 - accuracy: 0.9730 - val_loss: 0.1725 - val_accuracy: 0.9530
Epoch 17/100
59/59 [=====] - 1s 23ms/step - loss: 0.1202 - accuracy: 0.9725 - val_loss: 0.1786 - val_accuracy: 0.9488

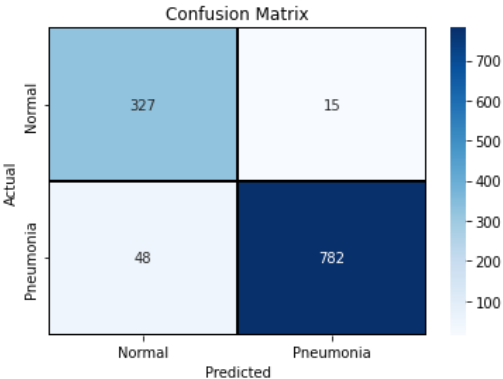
```
In [ ]: learn_curves(history2)
```

Final Training Accuracy: 0.972511351108551
Final Validation Accuracy: 0.948772668838501



```
In [ ]: evaluation(cnn2)
```

37/37 [=====] - 0s 8ms/step



37/37 [=====] - 0s 9ms/step - loss: 0.1885 - accuracy: 0.9462

[0.1885470449924469, 0.9462457299232483]

The l2 regularization helped, but we went a little light in our efforts.

So we are gonna do a couple of things. First, and this won't help with the fit, we will increase the complexity of the model and lower the learning rate to hopefully speed things up a bit.

We will introduce regularization on each conv block and include a dropout layer just before the classifier. The dropout is set at 0.50.

Model 3

```
In [ ]:
cnn3 = keras.Sequential()
# adding l2 to conv layers before pooling layers
cnn3.add(layers.Conv2D(8, 5, activation='relu', input_shape=(256, 256, 3)))
cnn3.add(layers.Conv2D(16, 3, activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn3.add(layers.MaxPooling2D(2, 2))

cnn3.add(layers.Conv2D(16, 3, activation='relu'))
cnn3.add(layers.Conv2D(32, 3, activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn3.add(layers.MaxPooling2D(2, 2))

cnn3.add(layers.Flatten())
cnn3.add(layers.Dense(64, activation='relu'))
cnn3.add(layers.Dropout(0.5)) #dropout before classifier
cnn3.add(layers.Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(1e-5)
cnn3.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn3.build(input_shape=(None, 256, 256, 3))
cnn3.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
=====		
conv2d_4 (Conv2D)	(None, 252, 252, 8)	608
conv2d_5 (Conv2D)	(None, 250, 250, 16)	1168
max_pooling2d_4 (MaxPooling 2D)	(None, 125, 125, 16)	0
conv2d_6 (Conv2D)	(None, 123, 123, 16)	2320
conv2d_7 (Conv2D)	(None, 121, 121, 32)	4640
max_pooling2d_5 (MaxPooling 2D)	(None, 60, 60, 32)	0
flatten_4 (Flatten)	(None, 115200)	0
dense_8 (Dense)	(None, 64)	7372864
dropout (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 1)	65
=====		
Total params: 7,381,665		
Trainable params: 7,381,665		
Non-trainable params: 0		

```
In [ ]:
history3 = cnn3.fit(X_train,
                    y_train,
                    epochs=100,
                    batch_size=64,
                    validation_split=0.20,
                    class_weight=class_weight,
                    callbacks=callbacks,
                    verbose=1)

Epoch 1/100
59/59 [=====] - 3s 56ms/step - loss: 0.9916 - accuracy: 0.5132 - val_loss: 0.9961 - val_accuracy: 0.6553
Epoch 2/100
59/59 [=====] - 2s 37ms/step - loss: 0.9617 - accuracy: 0.7003 - val_loss: 0.9441 - val_accuracy: 0.8709
Epoch 3/100
59/59 [=====] - 2s 37ms/step - loss: 0.9166 - accuracy: 0.7897 - val_loss: 0.8920 - val_accuracy: 0.8698
Epoch 4/100
59/59 [=====] - 2s 37ms/step - loss: 0.8543 - accuracy: 0.8244 - val_loss: 0.8128 - val_accuracy: 0.8773
Epoch 5/100
59/59 [=====] - 2s 37ms/step - loss: 0.7816 - accuracy: 0.8449 - val_loss: 0.7699 - val_accuracy: 0.8623
Epoch 6/100
59/59 [=====] - 2s 36ms/step - loss: 0.7194 - accuracy: 0.8529 - val_loss: 0.7073 - val_accuracy: 0.8709
Epoch 7/100
59/59 [=====] - 2s 37ms/step - loss: 0.6713 - accuracy: 0.8674 - val_loss: 0.6777 - val_accuracy: 0.8666
Epoch 8/100
59/59 [=====] - 2s 37ms/step - loss: 0.6442 - accuracy: 0.8788 - val_loss: 0.6569 - val_accuracy: 0.8719
Epoch 9/100
```

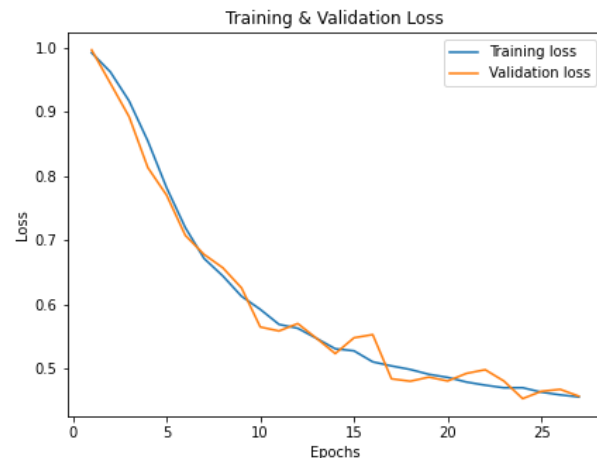
```

59/59 [=====] - 2s 37ms/step - loss: 0.6125 - accuracy: 0.8879 - val_loss: 0.6255 - val_accuracy: 0.8815
Epoch 10/100
59/59 [=====] - 2s 37ms/step - loss: 0.5921 - accuracy: 0.8906 - val_loss: 0.5647 - val_accuracy: 0.9157
Epoch 11/100
59/59 [=====] - 2s 36ms/step - loss: 0.5687 - accuracy: 0.8991 - val_loss: 0.5585 - val_accuracy: 0.9146
Epoch 12/100
59/59 [=====] - 2s 37ms/step - loss: 0.5627 - accuracy: 0.9007 - val_loss: 0.5700 - val_accuracy: 0.9072
Epoch 13/100
59/59 [=====] - 2s 37ms/step - loss: 0.5470 - accuracy: 0.9101 - val_loss: 0.5471 - val_accuracy: 0.9114
Epoch 14/100
59/59 [=====] - 2s 37ms/step - loss: 0.5308 - accuracy: 0.9162 - val_loss: 0.5233 - val_accuracy: 0.9264
Epoch 15/100
59/59 [=====] - 2s 37ms/step - loss: 0.5275 - accuracy: 0.9167 - val_loss: 0.5478 - val_accuracy: 0.9093
Epoch 16/100
59/59 [=====] - 2s 37ms/step - loss: 0.5104 - accuracy: 0.9213 - val_loss: 0.5528 - val_accuracy: 0.9050
Epoch 17/100
59/59 [=====] - 2s 37ms/step - loss: 0.5041 - accuracy: 0.9189 - val_loss: 0.4840 - val_accuracy: 0.9360
Epoch 18/100
59/59 [=====] - 2s 37ms/step - loss: 0.4986 - accuracy: 0.9223 - val_loss: 0.4802 - val_accuracy: 0.9349
Epoch 19/100
59/59 [=====] - 2s 37ms/step - loss: 0.4910 - accuracy: 0.9226 - val_loss: 0.4866 - val_accuracy: 0.9296
Epoch 20/100
59/59 [=====] - 2s 37ms/step - loss: 0.4862 - accuracy: 0.9261 - val_loss: 0.4806 - val_accuracy: 0.9306
Epoch 21/100
59/59 [=====] - 2s 37ms/step - loss: 0.4790 - accuracy: 0.9223 - val_loss: 0.4924 - val_accuracy: 0.9221
Epoch 22/100
59/59 [=====] - 2s 37ms/step - loss: 0.4741 - accuracy: 0.9218 - val_loss: 0.4982 - val_accuracy: 0.9232
Epoch 23/100
59/59 [=====] - 2s 37ms/step - loss: 0.4701 - accuracy: 0.9311 - val_loss: 0.4805 - val_accuracy: 0.9296
Epoch 24/100
59/59 [=====] - 2s 37ms/step - loss: 0.4701 - accuracy: 0.9287 - val_loss: 0.4531 - val_accuracy: 0.9402
Epoch 25/100
59/59 [=====] - 2s 37ms/step - loss: 0.4632 - accuracy: 0.9309 - val_loss: 0.4646 - val_accuracy: 0.9328
Epoch 26/100
59/59 [=====] - 2s 37ms/step - loss: 0.4590 - accuracy: 0.9274 - val_loss: 0.4676 - val_accuracy: 0.9328
Epoch 27/100
59/59 [=====] - 2s 37ms/step - loss: 0.4558 - accuracy: 0.9311 - val_loss: 0.4569 - val_accuracy: 0.9328

```

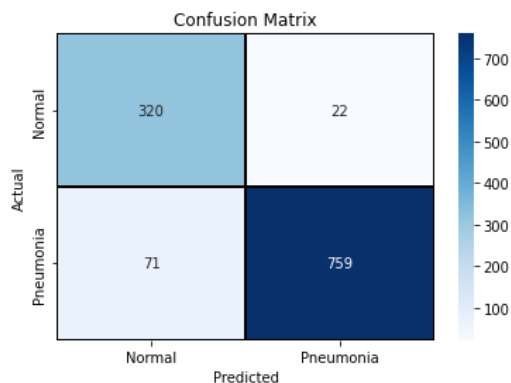
```
In [ ]: learn_curves(history3)
```

Final Training Accuracy: 0.9311448931694031
Final Validation Accuracy: 0.9327641129493713



```
In [ ]: evaluation(cnn3)
```

```
37/37 [=====] - 1s 11ms/step
```



```

37/37 [=====] - 0s 11ms/step - loss: 0.4639 - accuracy: 0.9206
[0.46389254927635193, 0.920648455619812]

```

That is looking a lot better. The fit looks fine now, but the model still seems a bit slow to converge. It's also rather noisy.

We will increase the complexity again, but limit our regularization to the dense layers, of which there are now two.

Another good way to keep overfitting in check is to introduce more data. For this we're adding a preprocessing layer that randomly tweaks the image contrast. Given the nature of X-ray imagery, this could be draw out new features or mask unimportant ones.

Model 4

```
In [ ]:
cnn4 = keras.Sequential()

#preprocessing layer to randomly alter image contrast for data augmentation
cnn4.add(layers.experimental.preprocessing.RandomContrast([0.2, 0.8]))

cnn4.add(layers.Conv2D(8, (5, 5), activation='relu', input_shape=(256, 256, 3)))
cnn4.add(layers.Conv2D(16, (3, 3), activation='relu'))
cnn4.add(layers.MaxPooling2D(2, 2))

cnn4.add(layers.Conv2D(16, (3, 3), activation='relu'))
cnn4.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn4.add(layers.MaxPooling2D(2, 2))

cnn4.add(layers.Flatten())
cnn4.add(layers.Dense(32, activation='relu'))
cnn4.add(layers.Dropout(0.4))
cnn4.add(layers.Dense(64, activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn4.add(layers.Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(1e-5)
cnn4.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn4.build(input_shape=(None, 256, 256, 3))
cnn4.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
=====		
random_contrast (RandomContrast)	(None, 256, 256, 3)	0
conv2d_8 (Conv2D)	(None, 252, 252, 8)	608
conv2d_9 (Conv2D)	(None, 250, 250, 16)	1168
max_pooling2d_6 (MaxPooling2D)	(None, 125, 125, 16)	0
conv2d_10 (Conv2D)	(None, 123, 123, 16)	2320
conv2d_11 (Conv2D)	(None, 121, 121, 32)	4640
max_pooling2d_7 (MaxPooling2D)	(None, 60, 60, 32)	0
flatten_5 (Flatten)	(None, 115200)	0
dense_10 (Dense)	(None, 32)	3686432
dropout_1 (Dropout)	(None, 32)	0
dense_11 (Dense)	(None, 64)	2112
dense_12 (Dense)	(None, 1)	65
=====		
Total params: 3,697,345		
Trainable params: 3,697,345		
Non-trainable params: 0		

```
In [ ]:
history4 = cnn4.fit(X_train,
                    y_train,
                    epochs=100,
                    batch_size=64,
                    validation_split=0.20,
                    class_weight=class_weight,
                    callbacks=callbacks,
                    verbose=1)

Epoch 1/100
59/59 [=====] - 17s 290ms/step - loss: 1.0992 - accuracy: 0.6015 - val_loss: 1.0720 - val_accuracy: 0.8762
Epoch 2/100
59/59 [=====] - 17s 283ms/step - loss: 1.0501 - accuracy: 0.7083 - val_loss: 1.0190 - val_accuracy: 0.8890
Epoch 3/100
59/59 [=====] - 17s 282ms/step - loss: 0.9812 - accuracy: 0.7772 - val_loss: 0.9356 - val_accuracy: 0.8965
Epoch 4/100
59/59 [=====] - 17s 283ms/step - loss: 0.9176 - accuracy: 0.8121 - val_loss: 0.8773 - val_accuracy: 0.8954
Epoch 5/100
59/59 [=====] - 17s 282ms/step - loss: 0.8663 - accuracy: 0.8300 - val_loss: 0.8100 - val_accuracy: 0.9061
Epoch 6/100
59/59 [=====] - 17s 280ms/step - loss: 0.8245 - accuracy: 0.8455 - val_loss: 0.7763 - val_accuracy: 0.9072
Epoch 7/100
59/59 [=====] - 17s 281ms/step - loss: 0.7988 - accuracy: 0.8529 - val_loss: 0.7153 - val_accuracy: 0.9125
Epoch 8/100
59/59 [=====] - 17s 283ms/step - loss: 0.7689 - accuracy: 0.8703 - val_loss: 0.6968 - val_accuracy: 0.9210
Epoch 9/100
59/59 [=====] - 17s 282ms/step - loss: 0.7448 - accuracy: 0.8804 - val_loss: 0.6656 - val_accuracy: 0.9072
Epoch 10/100
```

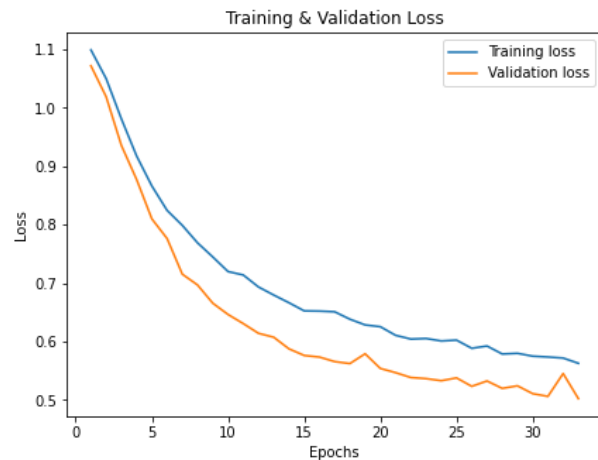
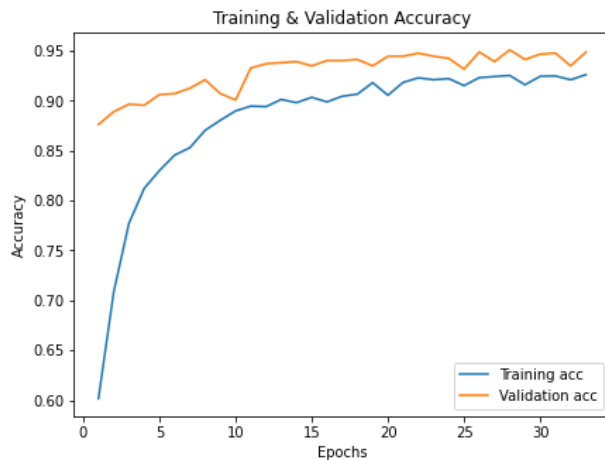
```

59/59 [=====] - 17s 284ms/step - loss: 0.7198 - accuracy: 0.8898 - val_loss: 0.6463 - val_accuracy: 0.9007
Epoch 11/100
59/59 [=====] - 17s 281ms/step - loss: 0.7139 - accuracy: 0.8946 - val_loss: 0.6307 - val_accuracy: 0.9328
Epoch 12/100
59/59 [=====] - 17s 282ms/step - loss: 0.6934 - accuracy: 0.8940 - val_loss: 0.6141 - val_accuracy: 0.9370
Epoch 13/100
59/59 [=====] - 17s 282ms/step - loss: 0.6796 - accuracy: 0.9013 - val_loss: 0.6074 - val_accuracy: 0.9381
Epoch 14/100
59/59 [=====] - 17s 281ms/step - loss: 0.6664 - accuracy: 0.8981 - val_loss: 0.5875 - val_accuracy: 0.9392
Epoch 15/100
59/59 [=====] - 17s 280ms/step - loss: 0.6525 - accuracy: 0.9034 - val_loss: 0.5762 - val_accuracy: 0.9349
Epoch 16/100
59/59 [=====] - 17s 281ms/step - loss: 0.6521 - accuracy: 0.8989 - val_loss: 0.5734 - val_accuracy: 0.9402
Epoch 17/100
59/59 [=====] - 17s 281ms/step - loss: 0.6509 - accuracy: 0.9045 - val_loss: 0.5656 - val_accuracy: 0.9402
Epoch 18/100
59/59 [=====] - 17s 280ms/step - loss: 0.6382 - accuracy: 0.9066 - val_loss: 0.5623 - val_accuracy: 0.9413
Epoch 19/100
59/59 [=====] - 17s 281ms/step - loss: 0.6285 - accuracy: 0.9181 - val_loss: 0.5790 - val_accuracy: 0.9349
Epoch 20/100
59/59 [=====] - 17s 281ms/step - loss: 0.6254 - accuracy: 0.9055 - val_loss: 0.5539 - val_accuracy: 0.9445
Epoch 21/100
59/59 [=====] - 17s 283ms/step - loss: 0.6107 - accuracy: 0.9183 - val_loss: 0.5467 - val_accuracy: 0.9445
Epoch 22/100
59/59 [=====] - 17s 281ms/step - loss: 0.6043 - accuracy: 0.9229 - val_loss: 0.5384 - val_accuracy: 0.9477
Epoch 23/100
59/59 [=====] - 17s 280ms/step - loss: 0.6052 - accuracy: 0.9210 - val_loss: 0.5365 - val_accuracy: 0.9445
Epoch 24/100
59/59 [=====] - 17s 281ms/step - loss: 0.6010 - accuracy: 0.9221 - val_loss: 0.5330 - val_accuracy: 0.9424
Epoch 25/100
59/59 [=====] - 17s 280ms/step - loss: 0.6024 - accuracy: 0.9151 - val_loss: 0.5377 - val_accuracy: 0.9317
Epoch 26/100
59/59 [=====] - 17s 281ms/step - loss: 0.5884 - accuracy: 0.9231 - val_loss: 0.5233 - val_accuracy: 0.9488
Epoch 27/100
59/59 [=====] - 17s 283ms/step - loss: 0.5923 - accuracy: 0.9242 - val_loss: 0.5325 - val_accuracy: 0.9392
Epoch 28/100
59/59 [=====] - 16s 280ms/step - loss: 0.5787 - accuracy: 0.9253 - val_loss: 0.5198 - val_accuracy: 0.9509
Epoch 29/100
59/59 [=====] - 17s 281ms/step - loss: 0.5798 - accuracy: 0.9159 - val_loss: 0.5242 - val_accuracy: 0.9413
Epoch 30/100
59/59 [=====] - 17s 284ms/step - loss: 0.5750 - accuracy: 0.9245 - val_loss: 0.5108 - val_accuracy: 0.9466
Epoch 31/100
59/59 [=====] - 17s 281ms/step - loss: 0.5735 - accuracy: 0.9247 - val_loss: 0.5061 - val_accuracy: 0.9477
Epoch 32/100
59/59 [=====] - 17s 281ms/step - loss: 0.5717 - accuracy: 0.9210 - val_loss: 0.5452 - val_accuracy: 0.9349
Epoch 33/100
59/59 [=====] - 17s 282ms/step - loss: 0.5628 - accuracy: 0.9261 - val_loss: 0.5022 - val_accuracy: 0.9488

```

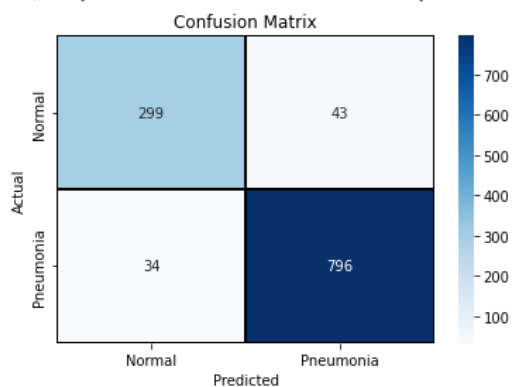
```
In [ ]: learn_curves(history4)
```

Final Training Accuracy: 0.926074206829071
Final Validation Accuracy: 0.948772668838501



```
In [ ]: evaluation(cnn4)
```

37/37 [=====] - 0s 10ms/step



```
37/37 [=====] - 0s 11ms/step - loss: 0.5236 - accuracy: 0.9343  
[0.5235985517501831, 0.9343003630638123]
```

Looks better still. But why stop here!

We're throwing caution to the wind and adding another conv block, increasing the number of filters on the dense layer (and nixing the smaller one).

Since complexity is increasing, we're beefing up our regularization with l2 regularizers and a dropout(0.5 again)

Model 5

```
In [ ]: cnn5 = keras.Sequential()

cnn5.add(layers.experimental.preprocessing.RandomContrast([0.2, 0.8]))

cnn5.add(layers.Conv2D(8, (5, 5), activation='relu', input_shape=(256, 256, 3)))
cnn5.add(layers.Conv2D(16, (3, 3), activation='relu'))
cnn5.add(layers.MaxPooling2D(2, 2))

cnn5.add(layers.Conv2D(16, (3, 3), activation='relu'))
cnn5.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn5.add(layers.MaxPooling2D(2, 2))

cnn5.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn5.add(layers.Conv2D(64, (3, 3), activation='relu',
                        kernel_regularizer=keras.regularizers.l2(0.01)))
cnn5.add(layers.MaxPooling2D(2, 2))

cnn5.add(layers.Flatten())
cnn5.add(layers.Dense(128, activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn5.add(layers.Dropout(0.5))
cnn5.add(layers.Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(1e-5)
cnn5.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn5.build(input_shape=(None, 256, 256, 3))
cnn5.summary()
```

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2
Model: "sequential"
```

Layer (type)	Output Shape	Param #
=====		
random_contrast (RandomContrast)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 252, 252, 8)	608
conv2d_1 (Conv2D)	(None, 250, 250, 16)	1168
max_pooling2d (MaxPooling2D)	(None, 125, 125, 16)	0
conv2d_2 (Conv2D)	(None, 123, 123, 16)	2320
conv2d_3 (Conv2D)	(None, 121, 121, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 60, 60, 32)	0
conv2d_4 (Conv2D)	(None, 58, 58, 64)	18496
conv2d_5 (Conv2D)	(None, 56, 56, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 64)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 128)	6422656
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129
=====		
Total params: 6,486,945		
Trainable params: 6,486,945		
Non-trainable params: 0		

```
In [ ]: history5 = cnn5.fit(X_train,
                           y_train,
                           epochs=100,
```



```
batch_size=64,  
validation_split=0.20,  
class_weight=class_weight,  
callbacks=callbacks,  
verbose=1)
```

Epoch 1/100

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip  
WARNING:tensorflow:Using a while_loop for converting Bitcast  
WARNING:tensorflow:Using a while_loop for converting Bitcast  
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2  
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter  
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2  
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2  
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip  
WARNING:tensorflow:Using a while_loop for converting Bitcast  
WARNING:tensorflow:Using a while_loop for converting Bitcast  
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2  
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter  
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2  
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2  
59/59 [=====] - 19s 295ms/step - loss: 3.6998 - accuracy: 0.5938 - val_loss: 3.5299 - val_accuracy: 0.7321  
Epoch 2/100  
59/59 [=====] - 17s 288ms/step - loss: 3.3324 - accuracy: 0.7732 - val_loss: 3.1528 - val_accuracy: 0.8719  
Epoch 3/100  
59/59 [=====] - 17s 290ms/step - loss: 2.9724 - accuracy: 0.8188 - val_loss: 2.8112 - val_accuracy: 0.8730  
Epoch 4/100  
59/59 [=====] - 17s 285ms/step - loss: 2.6624 - accuracy: 0.8484 - val_loss: 2.5107 - val_accuracy: 0.8975  
Epoch 5/100  
59/59 [=====] - 17s 284ms/step - loss: 2.4190 - accuracy: 0.8639 - val_loss: 2.2823 - val_accuracy: 0.8858  
Epoch 6/100  
59/59 [=====] - 17s 286ms/step - loss: 2.2286 - accuracy: 0.8732 - val_loss: 2.1367 - val_accuracy: 0.9007  
Epoch 7/100  
59/59 [=====] - 17s 287ms/step - loss: 2.0606 - accuracy: 0.8940 - val_loss: 1.9742 - val_accuracy: 0.9146  
Epoch 8/100  
59/59 [=====] - 17s 288ms/step - loss: 1.9349 - accuracy: 0.8906 - val_loss: 1.9320 - val_accuracy: 0.8741  
Epoch 9/100  
59/59 [=====] - 17s 285ms/step - loss: 1.8206 - accuracy: 0.8981 - val_loss: 1.7428 - val_accuracy: 0.9242  
Epoch 10/100  
59/59 [=====] - 17s 288ms/step - loss: 1.7153 - accuracy: 0.9047 - val_loss: 1.6554 - val_accuracy: 0.9253  
Epoch 11/100  
59/59 [=====] - 17s 285ms/step - loss: 1.6405 - accuracy: 0.9071 - val_loss: 1.5670 - val_accuracy: 0.9338  
Epoch 12/100  
59/59 [=====] - 17s 286ms/step - loss: 1.5610 - accuracy: 0.9093 - val_loss: 1.5062 - val_accuracy: 0.9285  
Epoch 13/100  
59/59 [=====] - 17s 286ms/step - loss: 1.4940 - accuracy: 0.9122 - val_loss: 1.4513 - val_accuracy: 0.9317  
Epoch 14/100  
59/59 [=====] - 17s 286ms/step - loss: 1.4408 - accuracy: 0.9133 - val_loss: 1.4035 - val_accuracy: 0.9317  
Epoch 15/100  
59/59 [=====] - 17s 287ms/step - loss: 1.3840 - accuracy: 0.9223 - val_loss: 1.3383 - val_accuracy: 0.9370  
Epoch 16/100  
59/59 [=====] - 17s 285ms/step - loss: 1.3440 - accuracy: 0.9202 - val_loss: 1.2978 - val_accuracy: 0.9402  
Epoch 17/100  
59/59 [=====] - 17s 286ms/step - loss: 1.2987 - accuracy: 0.9183 - val_loss: 1.2832 - val_accuracy: 0.9317  
Epoch 18/100  
59/59 [=====] - 17s 288ms/step - loss: 1.2702 - accuracy: 0.9210 - val_loss: 1.2601 - val_accuracy: 0.9296  
Epoch 19/100  
59/59 [=====] - 17s 288ms/step - loss: 1.2389 - accuracy: 0.9186 - val_loss: 1.1998 - val_accuracy: 0.9274  
Epoch 20/100  
59/59 [=====] - 17s 288ms/step - loss: 1.2100 - accuracy: 0.9279 - val_loss: 1.1680 - val_accuracy: 0.9370  
Epoch 21/100  
59/59 [=====] - 17s 288ms/step - loss: 1.1767 - accuracy: 0.9274 - val_loss: 1.1506 - val_accuracy: 0.9434  
Epoch 22/100  
59/59 [=====] - 17s 289ms/step - loss: 1.1574 - accuracy: 0.9247 - val_loss: 1.1213 - val_accuracy: 0.9402  
Epoch 23/100  
59/59 [=====] - 17s 289ms/step - loss: 1.1420 - accuracy: 0.9242 - val_loss: 1.1008 - val_accuracy: 0.9413  
Epoch 24/100  
59/59 [=====] - 17s 288ms/step - loss: 1.1187 - accuracy: 0.9293 - val_loss: 1.0774 - val_accuracy: 0.9424  
Epoch 25/100  
59/59 [=====] - 17s 288ms/step - loss: 1.0993 - accuracy: 0.9261 - val_loss: 1.0711 - val_accuracy: 0.9445  
Epoch 26/100  
59/59 [=====] - 17s 290ms/step - loss: 1.0733 - accuracy: 0.9330 - val_loss: 1.0460 - val_accuracy: 0.9434  
Epoch 27/100  
59/59 [=====] - 17s 288ms/step - loss: 1.0568 - accuracy: 0.9376 - val_loss: 1.0656 - val_accuracy: 0.9296  
Epoch 28/100  
59/59 [=====] - 17s 288ms/step - loss: 1.0422 - accuracy: 0.9357 - val_loss: 1.0410 - val_accuracy: 0.9328  
Epoch 29/100  
59/59 [=====] - 17s 288ms/step - loss: 1.0287 - accuracy: 0.9365 - val_loss: 1.0022 - val_accuracy: 0.9392  
Epoch 30/100  
59/59 [=====] - 17s 289ms/step - loss: 1.0199 - accuracy: 0.9349 - val_loss: 0.9960 - val_accuracy: 0.9434  
Epoch 31/100  
59/59 [=====] - 17s 287ms/step - loss: 1.0022 - accuracy: 0.9341 - val_loss: 0.9917 - val_accuracy: 0.9413  
Epoch 32/100  
59/59 [=====] - 17s 291ms/step - loss: 0.9889 - accuracy: 0.9381 - val_loss: 0.9631 - val_accuracy: 0.9413  
Epoch 33/100  
59/59 [=====] - 17s 290ms/step - loss: 0.9786 - accuracy: 0.9341 - val_loss: 0.9499 - val_accuracy: 0.9434  
Epoch 34/100  
59/59 [=====] - 17s 287ms/step - loss: 0.9647 - accuracy: 0.9370 - val_loss: 0.9811 - val_accuracy: 0.9296  
Epoch 35/100  
59/59 [=====] - 17s 287ms/step - loss: 0.9510 - accuracy: 0.9338 - val_loss: 0.9295 - val_accuracy: 0.9434  
Epoch 36/100  
59/59 [=====] - 17s 288ms/step - loss: 0.9384 - accuracy: 0.9378 - val_loss: 0.9236 - val_accuracy: 0.9456  
Epoch 37/100  
59/59 [=====] - 17s 288ms/step - loss: 0.9276 - accuracy: 0.9346 - val_loss: 0.9298 - val_accuracy: 0.9392  
Epoch 38/100  
59/59 [=====] - 17s 285ms/step - loss: 0.9122 - accuracy: 0.9429 - val_loss: 0.8969 - val_accuracy: 0.9424  
Epoch 39/100  
59/59 [=====] - 17s 287ms/step - loss: 0.9073 - accuracy: 0.9381 - val_loss: 0.8878 - val_accuracy: 0.9456
```

```

Epoch 40/100
59/59 [=====] - 17s 287ms/step - loss: 0.9003 - accuracy: 0.9359 - val_loss: 0.8963 - val_accuracy: 0.9381
Epoch 41/100
59/59 [=====] - 17s 289ms/step - loss: 0.8830 - accuracy: 0.9416 - val_loss: 0.8798 - val_accuracy: 0.9434
Epoch 42/100
59/59 [=====] - 17s 292ms/step - loss: 0.8769 - accuracy: 0.9413 - val_loss: 0.8668 - val_accuracy: 0.9456
Epoch 43/100
59/59 [=====] - 17s 289ms/step - loss: 0.8780 - accuracy: 0.9376 - val_loss: 0.8490 - val_accuracy: 0.9456
Epoch 44/100
59/59 [=====] - 17s 296ms/step - loss: 0.8697 - accuracy: 0.9357 - val_loss: 0.8972 - val_accuracy: 0.9253
Epoch 45/100
59/59 [=====] - 17s 290ms/step - loss: 0.8504 - accuracy: 0.9402 - val_loss: 0.8391 - val_accuracy: 0.9434
Epoch 46/100
59/59 [=====] - 17s 290ms/step - loss: 0.8396 - accuracy: 0.9400 - val_loss: 0.8285 - val_accuracy: 0.9466
Epoch 47/100
59/59 [=====] - 17s 290ms/step - loss: 0.8233 - accuracy: 0.9445 - val_loss: 0.8136 - val_accuracy: 0.9498
Epoch 48/100
59/59 [=====] - 17s 290ms/step - loss: 0.8246 - accuracy: 0.9416 - val_loss: 0.8109 - val_accuracy: 0.9466
Epoch 49/100
59/59 [=====] - 17s 291ms/step - loss: 0.8176 - accuracy: 0.9386 - val_loss: 0.8040 - val_accuracy: 0.9488
Epoch 50/100
59/59 [=====] - 17s 289ms/step - loss: 0.8013 - accuracy: 0.9461 - val_loss: 0.8094 - val_accuracy: 0.9402
Epoch 51/100
59/59 [=====] - 17s 292ms/step - loss: 0.7945 - accuracy: 0.9426 - val_loss: 0.7832 - val_accuracy: 0.9488
Epoch 52/100
59/59 [=====] - 17s 291ms/step - loss: 0.7923 - accuracy: 0.9450 - val_loss: 0.8070 - val_accuracy: 0.9360
Epoch 53/100
59/59 [=====] - 17s 290ms/step - loss: 0.7809 - accuracy: 0.9445 - val_loss: 0.7703 - val_accuracy: 0.9488
Epoch 54/100
59/59 [=====] - 17s 290ms/step - loss: 0.7752 - accuracy: 0.9429 - val_loss: 0.7799 - val_accuracy: 0.9402
Epoch 55/100
59/59 [=====] - 17s 288ms/step - loss: 0.7718 - accuracy: 0.9432 - val_loss: 0.8015 - val_accuracy: 0.9338
Epoch 56/100
59/59 [=====] - 17s 288ms/step - loss: 0.7611 - accuracy: 0.9434 - val_loss: 0.7531 - val_accuracy: 0.9488
Epoch 57/100
59/59 [=====] - 17s 287ms/step - loss: 0.7503 - accuracy: 0.9472 - val_loss: 0.7698 - val_accuracy: 0.9381
Epoch 58/100
59/59 [=====] - 17s 288ms/step - loss: 0.7415 - accuracy: 0.9434 - val_loss: 0.7371 - val_accuracy: 0.9498
Epoch 59/100
59/59 [=====] - 17s 287ms/step - loss: 0.7372 - accuracy: 0.9469 - val_loss: 0.7394 - val_accuracy: 0.9381
Epoch 60/100
59/59 [=====] - 17s 289ms/step - loss: 0.7301 - accuracy: 0.9453 - val_loss: 0.7305 - val_accuracy: 0.9434
Epoch 61/100
59/59 [=====] - 17s 286ms/step - loss: 0.7260 - accuracy: 0.9453 - val_loss: 0.7258 - val_accuracy: 0.9434
Epoch 62/100
59/59 [=====] - 17s 289ms/step - loss: 0.7170 - accuracy: 0.9424 - val_loss: 0.7228 - val_accuracy: 0.9434
Epoch 63/100
59/59 [=====] - 17s 288ms/step - loss: 0.7032 - accuracy: 0.9490 - val_loss: 0.7013 - val_accuracy: 0.9530
Epoch 64/100
59/59 [=====] - 17s 287ms/step - loss: 0.6983 - accuracy: 0.9488 - val_loss: 0.6929 - val_accuracy: 0.9498
Epoch 65/100
59/59 [=====] - 17s 290ms/step - loss: 0.6950 - accuracy: 0.9485 - val_loss: 0.7069 - val_accuracy: 0.9434
Epoch 66/100
59/59 [=====] - 17s 286ms/step - loss: 0.6860 - accuracy: 0.9490 - val_loss: 0.6869 - val_accuracy: 0.9509
Epoch 67/100
59/59 [=====] - 17s 287ms/step - loss: 0.6812 - accuracy: 0.9480 - val_loss: 0.7071 - val_accuracy: 0.9413
Epoch 68/100
59/59 [=====] - 17s 288ms/step - loss: 0.6926 - accuracy: 0.9410 - val_loss: 0.6715 - val_accuracy: 0.9509
Epoch 69/100
59/59 [=====] - 17s 286ms/step - loss: 0.6653 - accuracy: 0.9509 - val_loss: 0.6648 - val_accuracy: 0.9509
Epoch 70/100
59/59 [=====] - 17s 287ms/step - loss: 0.6604 - accuracy: 0.9485 - val_loss: 0.6710 - val_accuracy: 0.9477
Epoch 71/100
59/59 [=====] - 17s 288ms/step - loss: 0.6574 - accuracy: 0.9509 - val_loss: 0.6643 - val_accuracy: 0.9488
Epoch 72/100
59/59 [=====] - 17s 286ms/step - loss: 0.6536 - accuracy: 0.9501 - val_loss: 0.6586 - val_accuracy: 0.9498
Epoch 73/100
59/59 [=====] - 17s 286ms/step - loss: 0.6412 - accuracy: 0.9517 - val_loss: 0.6520 - val_accuracy: 0.9530
Epoch 74/100
59/59 [=====] - 17s 287ms/step - loss: 0.6447 - accuracy: 0.9488 - val_loss: 0.6367 - val_accuracy: 0.9520
Epoch 75/100
59/59 [=====] - 17s 287ms/step - loss: 0.6395 - accuracy: 0.9488 - val_loss: 0.6446 - val_accuracy: 0.9498
Epoch 76/100
59/59 [=====] - 17s 287ms/step - loss: 0.6365 - accuracy: 0.9482 - val_loss: 0.6299 - val_accuracy: 0.9509
Epoch 77/100
59/59 [=====] - 17s 287ms/step - loss: 0.6264 - accuracy: 0.9522 - val_loss: 0.6295 - val_accuracy: 0.9530
Epoch 78/100
59/59 [=====] - 17s 287ms/step - loss: 0.6193 - accuracy: 0.9533 - val_loss: 0.6281 - val_accuracy: 0.9520

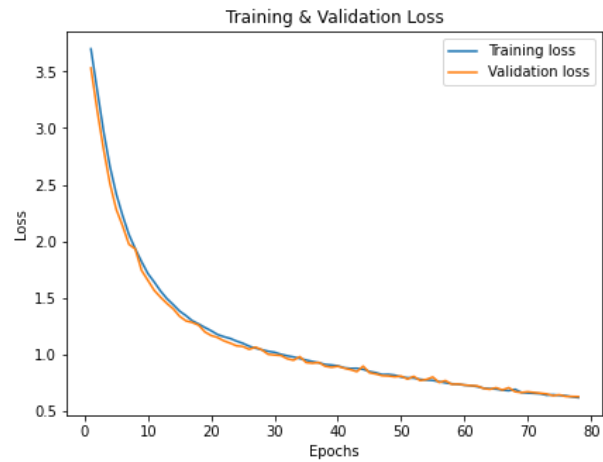
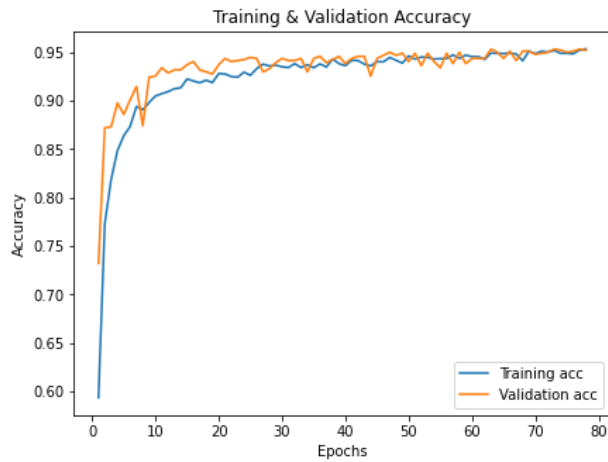
```

```
In [ ]: learn_curves(history5)
```

```

Final Training Accuracy: 0.9532959461212158
Final Validation Accuracy: 0.9519743919372559

```



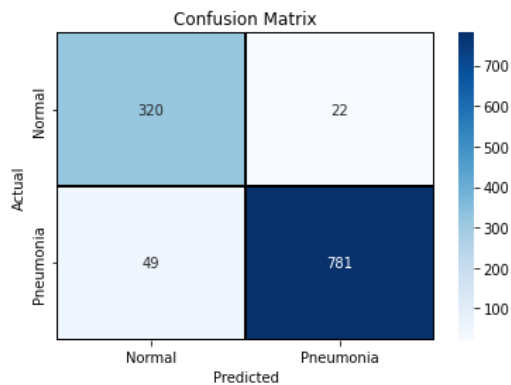
```
In [ ]:
def evaluation(model):
    predictions = (model.predict(X_test) > 0.5)*1
    cm = confusion_matrix(y_test, predictions)

    df_cm = pd.DataFrame(cm.T, index=['Pneumonia', 'Normal'],
                        columns=['Pneumonia', 'Normal'])
    sns.heatmap(df_cm.T, annot=True, fmt='d', cmap='Blues',
                linecolor='black', linewidths=1)
    plt.title('Confusion Matrix')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.ylim(0, 2)
    plt.xlim(2, 0)
    plt.show()

    print(model.evaluate(X_test, y_test))
```

```
In [ ]:
evaluation(cnn5)
```

37/37 [=====] - 1s 11ms/step



37/37 [=====] - 0s 11ms/step - loss: 0.6450 - accuracy: 0.9394
[0.6450021862983704, 0.9394198060035706]

That created a few more false negatives than I'm comfortable with. Generally speaking, the balance between bias and variance is going to be situation-dependent. In some instances, having more false positives is preferable to false negatives, such as here.

For medical situations, it's better to treat an illness that isn't there than to miss an illness that actually is there. So we want to minimize our false negatives as much as we can.

Model 6

```
In [ ]:
cnn6 = keras.Sequential()

cnn6.add(layers.experimental.preprocessing.RandomContrast([0.2, 0.8]))

cnn6.add(layers.Conv2D(8, (5, 5), activation='relu', input_shape=(256, 256, 3)))
cnn6.add(layers.Conv2D(16, (3, 3), activation='relu'))
cnn6.add(layers.MaxPooling2D(2, 2))

cnn6.add(layers.Conv2D(16, (3, 3), activation='relu'))
cnn6.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn6.add(layers.MaxPooling2D(2, 2))

cnn6.add(layers.Conv2D(64, (3, 3), activation='relu',
                        kernel_regularizer=keras.regularizers.l2(0.01)))
cnn6.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn6.add(layers.Dropout(0.5))
```

```

cnn6.add(layers.MaxPooling2D(2, 2))

cnn6.add(layers.Flatten()),
cnn6.add(layers.Dense(256, activation='relu')),
cnn6.add(layers.Dropout(0.5))
cnn6.add(layers.Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(1e-5)
cnn6.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn6.build(input_shape=(None, 256, 256, 3))
cnn6.summary()

```

Out[]: (None,)

Out[]: (None,)

Model: "sequential_1"

Layer (type)	Output Shape	Param #
random_contrast_1 (RandomContrast)	(None, 256, 256, 3)	0
conv2d_6 (Conv2D)	(None, 252, 252, 8)	608
conv2d_7 (Conv2D)	(None, 250, 250, 16)	1168
max_pooling2d_3 (MaxPooling2D)	(None, 125, 125, 16)	0
conv2d_8 (Conv2D)	(None, 123, 123, 16)	2320
conv2d_9 (Conv2D)	(None, 121, 121, 32)	4640
max_pooling2d_4 (MaxPooling2D)	(None, 60, 60, 32)	0
conv2d_10 (Conv2D)	(None, 58, 58, 64)	18496
conv2d_11 (Conv2D)	(None, 56, 56, 128)	73856
dropout_1 (Dropout)	(None, 56, 56, 128)	0
max_pooling2d_5 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten_1 (Flatten)	(None, 100352)	0
dense_2 (Dense)	(None, 256)	25690368
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 1)	257
Total params: 25,791,713		
Trainable params: 25,791,713		
Non-trainable params: 0		

In []:

```

opt = keras.optimizers.Adam(1e-5)

cnn6.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn6.build(input_shape=(None, 256, 256, 3))
cnn6.summary()

```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
random_contrast_1 (RandomContrast)	(None, 256, 256, 3)	0
conv2d_6 (Conv2D)	(None, 252, 252, 8)	608
conv2d_7 (Conv2D)	(None, 250, 250, 16)	1168
max_pooling2d_3 (MaxPooling2D)	(None, 125, 125, 16)	0
conv2d_8 (Conv2D)	(None, 123, 123, 16)	2320
conv2d_9 (Conv2D)	(None, 121, 121, 32)	4640
max_pooling2d_4 (MaxPooling2D)	(None, 60, 60, 32)	0
conv2d_10 (Conv2D)	(None, 58, 58, 64)	18496
conv2d_11 (Conv2D)	(None, 56, 56, 128)	73856
dropout_1 (Dropout)	(None, 56, 56, 128)	0
max_pooling2d_5 (MaxPooling2D)	(None, 28, 28, 128)	0

flatten_1 (Flatten)	(None, 100352)	0
dense_2 (Dense)	(None, 256)	25690368
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 1)	257

=====
Total params: 25,791,713
Trainable params: 25,791,713
Non-trainable params: 0

In []:

```
history6 = cnn6.fit(X_train,
                    y_train,
                    epochs=100,
                    batch_size=64,
                    validation_split=0.20,
                    class_weight=class_weight,
                    callbacks=callbacks,
                    verbose=1)
```

Epoch 1/100

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2
59/59 [=====] - 20s 300ms/step - loss: 1.0473 - accuracy: 0.6976 - val_loss: 1.0242 - val_accuracy: 0.8751
Epoch 2/100
59/59 [=====] - 17s 292ms/step - loss: 0.8714 - accuracy: 0.8452 - val_loss: 0.9415 - val_accuracy: 0.8420
Epoch 3/100
59/59 [=====] - 17s 293ms/step - loss: 0.7487 - accuracy: 0.8796 - val_loss: 0.8446 - val_accuracy: 0.8933
Epoch 4/100
59/59 [=====] - 17s 297ms/step - loss: 0.6876 - accuracy: 0.8965 - val_loss: 0.7458 - val_accuracy: 0.9253
Epoch 5/100
59/59 [=====] - 17s 291ms/step - loss: 0.6589 - accuracy: 0.8997 - val_loss: 0.7200 - val_accuracy: 0.9360
Epoch 6/100
59/59 [=====] - 17s 291ms/step - loss: 0.6246 - accuracy: 0.9149 - val_loss: 0.6911 - val_accuracy: 0.9413
Epoch 7/100
59/59 [=====] - 17s 291ms/step - loss: 0.5995 - accuracy: 0.9205 - val_loss: 0.6570 - val_accuracy: 0.9392
Epoch 8/100
59/59 [=====] - 17s 292ms/step - loss: 0.5968 - accuracy: 0.9178 - val_loss: 0.7072 - val_accuracy: 0.9157
Epoch 9/100
59/59 [=====] - 17s 290ms/step - loss: 0.5787 - accuracy: 0.9263 - val_loss: 0.6504 - val_accuracy: 0.9392
Epoch 10/100
59/59 [=====] - 17s 290ms/step - loss: 0.5735 - accuracy: 0.9205 - val_loss: 0.6638 - val_accuracy: 0.9274
Epoch 11/100
59/59 [=====] - 17s 291ms/step - loss: 0.5659 - accuracy: 0.9226 - val_loss: 0.6214 - val_accuracy: 0.9424
Epoch 12/100
59/59 [=====] - 17s 290ms/step - loss: 0.5616 - accuracy: 0.9250 - val_loss: 0.6221 - val_accuracy: 0.9360
Epoch 13/100
59/59 [=====] - 17s 290ms/step - loss: 0.5356 - accuracy: 0.9359 - val_loss: 0.5969 - val_accuracy: 0.9413
Epoch 14/100
59/59 [=====] - 17s 292ms/step - loss: 0.5342 - accuracy: 0.9351 - val_loss: 0.5804 - val_accuracy: 0.9466
Epoch 15/100
59/59 [=====] - 17s 290ms/step - loss: 0.5247 - accuracy: 0.9333 - val_loss: 0.5752 - val_accuracy: 0.9456
Epoch 16/100
59/59 [=====] - 17s 290ms/step - loss: 0.5146 - accuracy: 0.9367 - val_loss: 0.5700 - val_accuracy: 0.9445
Epoch 17/100
59/59 [=====] - 17s 292ms/step - loss: 0.5053 - accuracy: 0.9376 - val_loss: 0.5674 - val_accuracy: 0.9381
Epoch 18/100
59/59 [=====] - 17s 291ms/step - loss: 0.4956 - accuracy: 0.9397 - val_loss: 0.5670 - val_accuracy: 0.9381
Epoch 19/100
59/59 [=====] - 17s 290ms/step - loss: 0.4877 - accuracy: 0.9440 - val_loss: 0.5584 - val_accuracy: 0.9370
Epoch 20/100
59/59 [=====] - 17s 290ms/step - loss: 0.4855 - accuracy: 0.9394 - val_loss: 0.5489 - val_accuracy: 0.9456
Epoch 21/100
59/59 [=====] - 17s 292ms/step - loss: 0.4812 - accuracy: 0.9397 - val_loss: 0.5806 - val_accuracy: 0.9306
Epoch 22/100
59/59 [=====] - 17s 291ms/step - loss: 0.4754 - accuracy: 0.9432 - val_loss: 0.5786 - val_accuracy: 0.9306
Epoch 23/100
59/59 [=====] - 17s 293ms/step - loss: 0.4611 - accuracy: 0.9448 - val_loss: 0.5404 - val_accuracy: 0.9381
Epoch 24/100
59/59 [=====] - 17s 294ms/step - loss: 0.4603 - accuracy: 0.9477 - val_loss: 0.5506 - val_accuracy: 0.9349
Epoch 25/100
59/59 [=====] - 17s 291ms/step - loss: 0.4543 - accuracy: 0.9466 - val_loss: 0.5192 - val_accuracy: 0.9434
Epoch 26/100
59/59 [=====] - 17s 293ms/step - loss: 0.4486 - accuracy: 0.9448 - val_loss: 0.4948 - val_accuracy: 0.9520
Epoch 27/100
59/59 [=====] - 17s 291ms/step - loss: 0.4345 - accuracy: 0.9546 - val_loss: 0.4991 - val_accuracy: 0.9456
Epoch 28/100
59/59 [=====] - 17s 292ms/step - loss: 0.4298 - accuracy: 0.9496 - val_loss: 0.4829 - val_accuracy: 0.9488
Epoch 29/100
59/59 [=====] - 17s 291ms/step - loss: 0.4328 - accuracy: 0.9504 - val_loss: 0.4796 - val_accuracy: 0.9530
Epoch 30/100
```

```

59/59 [=====] - 17s 292ms/step - loss: 0.4207 - accuracy: 0.9509 - val_loss: 0.5042 - val_accuracy: 0.9413
Epoch 31/100
59/59 [=====] - 17s 293ms/step - loss: 0.4281 - accuracy: 0.9445 - val_loss: 0.4674 - val_accuracy: 0.9488
Epoch 32/100
59/59 [=====] - 17s 291ms/step - loss: 0.4200 - accuracy: 0.9512 - val_loss: 0.4767 - val_accuracy: 0.9456
Epoch 33/100
59/59 [=====] - 17s 292ms/step - loss: 0.4081 - accuracy: 0.9530 - val_loss: 0.4819 - val_accuracy: 0.9456
Epoch 34/100
59/59 [=====] - 17s 292ms/step - loss: 0.4079 - accuracy: 0.9512 - val_loss: 0.4814 - val_accuracy: 0.9456
Epoch 35/100
59/59 [=====] - 17s 295ms/step - loss: 0.3946 - accuracy: 0.9546 - val_loss: 0.4806 - val_accuracy: 0.9424
Epoch 36/100
59/59 [=====] - 17s 291ms/step - loss: 0.3936 - accuracy: 0.9533 - val_loss: 0.4583 - val_accuracy: 0.9456
Epoch 37/100
59/59 [=====] - 17s 291ms/step - loss: 0.3885 - accuracy: 0.9581 - val_loss: 0.4442 - val_accuracy: 0.9488
Epoch 38/100
59/59 [=====] - 17s 291ms/step - loss: 0.3915 - accuracy: 0.9541 - val_loss: 0.4713 - val_accuracy: 0.9402
Epoch 39/100
59/59 [=====] - 17s 292ms/step - loss: 0.3849 - accuracy: 0.9541 - val_loss: 0.4420 - val_accuracy: 0.9498
Epoch 40/100
59/59 [=====] - 17s 292ms/step - loss: 0.3829 - accuracy: 0.9538 - val_loss: 0.4344 - val_accuracy: 0.9498
Epoch 41/100
59/59 [=====] - 17s 291ms/step - loss: 0.3804 - accuracy: 0.9586 - val_loss: 0.4498 - val_accuracy: 0.9466

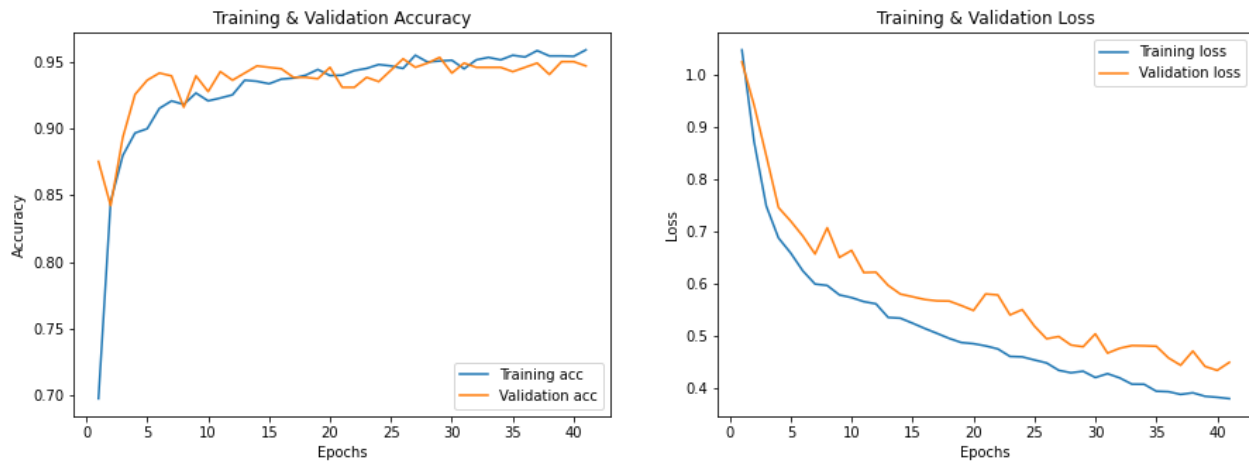
```

```
In [ ]: learn_curves(history6)
```

```

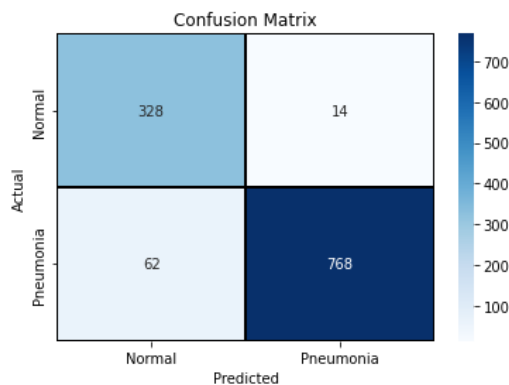
Final Training Accuracy: 0.9586336016654968
Final Validation Accuracy: 0.9466382265090942

```



```
In [ ]: evaluation(cnn6)
```

```
37/37 [=====] - 1s 11ms/step
```



```

37/37 [=====] - 0s 12ms/step - loss: 0.4628 - accuracy: 0.9352
[0.4627733528614044, 0.935153603553772]

```

FNs went up this time.

```
In [ ]: keras.backend.clear_session()
```

```
In [ ]: callbacks = EarlyStopping(monitor='val_loss', mode='min', patience=10)
```

Model 7

Here we will increase number of filters in all layers, regularize the final conv layer and both dense layers, and dropout just before and after flatten.

```
In [ ]:
cnn7 = keras.Sequential()

cnn7.add(layers.experimental.preprocessing.RandomContrast([0.2, 0.8]))
```

```
# doubling the number of filters at all layers
cnn7.add(layers.Conv2D(16, (5, 5), activation='relu', input_shape=(256, 256, 3)))
cnn7.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn7.add(layers.MaxPooling2D(2, 2))

cnn7.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn7.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn7.add(layers.MaxPooling2D(2, 2))

cnn7.add(layers.Conv2D(128, (3, 3), activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn7.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn7.add(layers.Dropout(0.5))
cnn7.add(layers.MaxPooling2D(2, 2))

# heavy regularization on fully connected layers
cnn7.add(layers.Flatten())
cnn7.add(layers.Dense(256, activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn7.add(layers.Dense(512, activation='relu',
                      kernel_regularizer=keras.regularizers.l2(0.01)))
cnn7.add(layers.Dropout(0.5))
cnn7.add(layers.Dense(1, activation='sigmoid'))
```

```
In [ ]: opt = keras.optimizers.Adam(1e-5)

cnn7.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
cnn7.build(input_shape=(None, 256, 256, 3))
cnn7.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
=====		
random_contrast_2 (RandomContrast)	(None, 256, 256, 3)	0
conv2d_12 (Conv2D)	(None, 252, 252, 16)	1216
conv2d_13 (Conv2D)	(None, 250, 250, 32)	4640
max_pooling2d_6 (MaxPooling2D)	(None, 125, 125, 32)	0
conv2d_14 (Conv2D)	(None, 123, 123, 64)	18496
conv2d_15 (Conv2D)	(None, 121, 121, 64)	36928
max_pooling2d_7 (MaxPooling2D)	(None, 60, 60, 64)	0
conv2d_16 (Conv2D)	(None, 58, 58, 128)	73856
conv2d_17 (Conv2D)	(None, 56, 56, 128)	147584
dropout_3 (Dropout)	(None, 56, 56, 128)	0
max_pooling2d_8 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten_2 (Flatten)	(None, 100352)	0
dense_4 (Dense)	(None, 256)	25690368
dense_5 (Dense)	(None, 512)	131584
dropout_4 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 1)	513
=====		
Total params: 26,105,185		
Trainable params: 26,105,185		
Non-trainable params: 0		

```
In [ ]: history7 = cnn7.fit(X_train,
                           y_train,
                           epochs=100,
                           batch_size=64,
                           validation_split=0.20,
                           class_weight=class_weight,
                           callbacks=callbacks,
                           verbose=1)
```

Epoch 1/100

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
```

```
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting Bitcast
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformFullIntV2
WARNING:tensorflow:Using a while_loop for converting StatelessRandomGetKeyCounter
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2
59/59 [=====] - 23s 339ms/step - loss: 9.5272 - accuracy: 0.4825 - val_loss: 9.0064 - val_accuracy: 0.8677
Epoch 2/100
59/59 [=====] - 18s 312ms/step - loss: 8.4765 - accuracy: 0.7860 - val_loss: 8.1038 - val_accuracy: 0.7962
Epoch 3/100
59/59 [=====] - 18s 313ms/step - loss: 7.5799 - accuracy: 0.8511 - val_loss: 7.2930 - val_accuracy: 0.9072
Epoch 4/100
59/59 [=====] - 18s 313ms/step - loss: 6.9365 - accuracy: 0.8743 - val_loss: 6.7405 - val_accuracy: 0.9157
Epoch 5/100
59/59 [=====] - 18s 314ms/step - loss: 6.4224 - accuracy: 0.8965 - val_loss: 6.2832 - val_accuracy: 0.9306
Epoch 6/100
59/59 [=====] - 18s 313ms/step - loss: 6.0153 - accuracy: 0.9130 - val_loss: 5.9085 - val_accuracy: 0.9328
Epoch 7/100
59/59 [=====] - 19s 316ms/step - loss: 5.6813 - accuracy: 0.9066 - val_loss: 5.6078 - val_accuracy: 0.9381
Epoch 8/100
59/59 [=====] - 18s 313ms/step - loss: 5.3970 - accuracy: 0.9159 - val_loss: 5.3440 - val_accuracy: 0.9413
Epoch 9/100
59/59 [=====] - 18s 312ms/step - loss: 5.1731 - accuracy: 0.9133 - val_loss: 5.1243 - val_accuracy: 0.9424
Epoch 10/100
59/59 [=====] - 19s 314ms/step - loss: 4.9480 - accuracy: 0.9239 - val_loss: 4.9398 - val_accuracy: 0.9360
Epoch 11/100
59/59 [=====] - 18s 314ms/step - loss: 4.7837 - accuracy: 0.9242 - val_loss: 4.7498 - val_accuracy: 0.9381
Epoch 12/100
59/59 [=====] - 18s 314ms/step - loss: 4.6489 - accuracy: 0.9183 - val_loss: 4.6234 - val_accuracy: 0.9456
Epoch 13/100
59/59 [=====] - 18s 313ms/step - loss: 4.4907 - accuracy: 0.9293 - val_loss: 4.4818 - val_accuracy: 0.9456
Epoch 14/100
59/59 [=====] - 18s 313ms/step - loss: 4.3639 - accuracy: 0.9338 - val_loss: 4.3804 - val_accuracy: 0.9381
Epoch 15/100
59/59 [=====] - 19s 314ms/step - loss: 4.2529 - accuracy: 0.9381 - val_loss: 4.2510 - val_accuracy: 0.9477
Epoch 16/100
59/59 [=====] - 18s 312ms/step - loss: 4.1589 - accuracy: 0.9418 - val_loss: 4.1723 - val_accuracy: 0.9424
Epoch 17/100
59/59 [=====] - 19s 314ms/step - loss: 4.0655 - accuracy: 0.9373 - val_loss: 4.0719 - val_accuracy: 0.9466
Epoch 18/100
59/59 [=====] - 19s 314ms/step - loss: 3.9915 - accuracy: 0.9333 - val_loss: 3.9828 - val_accuracy: 0.9488
Epoch 19/100
59/59 [=====] - 19s 321ms/step - loss: 3.9000 - accuracy: 0.9381 - val_loss: 3.9131 - val_accuracy: 0.9477
Epoch 20/100
59/59 [=====] - 18s 313ms/step - loss: 3.8111 - accuracy: 0.9453 - val_loss: 3.8241 - val_accuracy: 0.9477
Epoch 21/100
59/59 [=====] - 18s 313ms/step - loss: 3.7464 - accuracy: 0.9418 - val_loss: 3.7526 - val_accuracy: 0.9530
Epoch 22/100
59/59 [=====] - 18s 314ms/step - loss: 3.6649 - accuracy: 0.9482 - val_loss: 3.7097 - val_accuracy: 0.9392
Epoch 23/100
59/59 [=====] - 18s 312ms/step - loss: 3.5990 - accuracy: 0.9469 - val_loss: 3.6178 - val_accuracy: 0.9477
Epoch 24/100
59/59 [=====] - 19s 316ms/step - loss: 3.5360 - accuracy: 0.9458 - val_loss: 3.5757 - val_accuracy: 0.9381
Epoch 25/100
59/59 [=====] - 18s 314ms/step - loss: 3.4635 - accuracy: 0.9472 - val_loss: 3.4783 - val_accuracy: 0.9488
Epoch 26/100
59/59 [=====] - 18s 313ms/step - loss: 3.4264 - accuracy: 0.9421 - val_loss: 3.4571 - val_accuracy: 0.9360
Epoch 27/100
59/59 [=====] - 18s 312ms/step - loss: 3.3521 - accuracy: 0.9480 - val_loss: 3.3729 - val_accuracy: 0.9488
Epoch 28/100
59/59 [=====] - 18s 313ms/step - loss: 3.2860 - accuracy: 0.9472 - val_loss: 3.3057 - val_accuracy: 0.9498
Epoch 29/100
59/59 [=====] - 18s 313ms/step - loss: 3.2273 - accuracy: 0.9493 - val_loss: 3.2596 - val_accuracy: 0.9520
Epoch 30/100
59/59 [=====] - 19s 314ms/step - loss: 3.1718 - accuracy: 0.9520 - val_loss: 3.2149 - val_accuracy: 0.9456
Epoch 31/100
59/59 [=====] - 18s 313ms/step - loss: 3.1127 - accuracy: 0.9522 - val_loss: 3.1335 - val_accuracy: 0.9498
Epoch 32/100
59/59 [=====] - 19s 314ms/step - loss: 3.0589 - accuracy: 0.9517 - val_loss: 3.0786 - val_accuracy: 0.9541
Epoch 33/100
59/59 [=====] - 18s 313ms/step - loss: 3.0086 - accuracy: 0.9517 - val_loss: 3.0461 - val_accuracy: 0.9477
Epoch 34/100
59/59 [=====] - 18s 313ms/step - loss: 2.9599 - accuracy: 0.9512 - val_loss: 2.9763 - val_accuracy: 0.9562
Epoch 35/100
59/59 [=====] - 19s 316ms/step - loss: 2.9037 - accuracy: 0.9541 - val_loss: 2.9351 - val_accuracy: 0.9573
Epoch 36/100
59/59 [=====] - 19s 317ms/step - loss: 2.8511 - accuracy: 0.9584 - val_loss: 2.8968 - val_accuracy: 0.9509
Epoch 37/100
59/59 [=====] - 18s 313ms/step - loss: 2.8067 - accuracy: 0.9536 - val_loss: 2.8388 - val_accuracy: 0.9530
Epoch 38/100
59/59 [=====] - 19s 317ms/step - loss: 2.7641 - accuracy: 0.9498 - val_loss: 2.7901 - val_accuracy: 0.9498
Epoch 39/100
59/59 [=====] - 18s 313ms/step - loss: 2.7117 - accuracy: 0.9536 - val_loss: 2.7420 - val_accuracy: 0.9562
Epoch 40/100
59/59 [=====] - 18s 313ms/step - loss: 2.6706 - accuracy: 0.9538 - val_loss: 2.7016 - val_accuracy: 0.9562
Epoch 41/100
59/59 [=====] - 19s 317ms/step - loss: 2.6206 - accuracy: 0.9560 - val_loss: 2.6587 - val_accuracy: 0.9541
Epoch 42/100
59/59 [=====] - 18s 313ms/step - loss: 2.5848 - accuracy: 0.9525 - val_loss: 2.6077 - val_accuracy: 0.9552
Epoch 43/100
59/59 [=====] - 18s 313ms/step - loss: 2.5311 - accuracy: 0.9552 - val_loss: 2.5625 - val_accuracy: 0.9530
Epoch 44/100
59/59 [=====] - 19s 314ms/step - loss: 2.4917 - accuracy: 0.9557 - val_loss: 2.5129 - val_accuracy: 0.9541
Epoch 45/100
59/59 [=====] - 18s 313ms/step - loss: 2.4465 - accuracy: 0.9578 - val_loss: 2.4909 - val_accuracy: 0.9541
Epoch 46/100
59/59 [=====] - 18s 313ms/step - loss: 2.4106 - accuracy: 0.9554 - val_loss: 2.4344 - val_accuracy: 0.9573
```


Epoch 47/100
59/59 [=====] - 18s 313ms/step - loss: 2.3684 - accuracy: 0.9581 - val_loss: 2.3942 - val_accuracy: 0.9562
Epoch 48/100
59/59 [=====] - 18s 313ms/step - loss: 2.3285 - accuracy: 0.9608 - val_loss: 2.4015 - val_accuracy: 0.9413
Epoch 49/100
59/59 [=====] - 18s 313ms/step - loss: 2.2852 - accuracy: 0.9589 - val_loss: 2.3275 - val_accuracy: 0.9584
Epoch 50/100
59/59 [=====] - 18s 312ms/step - loss: 2.2475 - accuracy: 0.9584 - val_loss: 2.2809 - val_accuracy: 0.9562
Epoch 51/100
59/59 [=====] - 18s 314ms/step - loss: 2.2087 - accuracy: 0.9624 - val_loss: 2.2420 - val_accuracy: 0.9573
Epoch 52/100
59/59 [=====] - 18s 313ms/step - loss: 2.1688 - accuracy: 0.9632 - val_loss: 2.2293 - val_accuracy: 0.9552
Epoch 53/100
59/59 [=====] - 19s 314ms/step - loss: 2.1367 - accuracy: 0.9618 - val_loss: 2.1752 - val_accuracy: 0.9605
Epoch 54/100
59/59 [=====] - 18s 313ms/step - loss: 2.0982 - accuracy: 0.9634 - val_loss: 2.1564 - val_accuracy: 0.9541
Epoch 55/100
59/59 [=====] - 18s 313ms/step - loss: 2.0662 - accuracy: 0.9608 - val_loss: 2.1262 - val_accuracy: 0.9530
Epoch 56/100
59/59 [=====] - 18s 313ms/step - loss: 2.0450 - accuracy: 0.9568 - val_loss: 2.1077 - val_accuracy: 0.9445
Epoch 57/100
59/59 [=====] - 19s 317ms/step - loss: 2.0078 - accuracy: 0.9605 - val_loss: 2.0668 - val_accuracy: 0.9466
Epoch 58/100
59/59 [=====] - 19s 314ms/step - loss: 1.9711 - accuracy: 0.9570 - val_loss: 2.0025 - val_accuracy: 0.9584
Epoch 59/100
59/59 [=====] - 18s 312ms/step - loss: 1.9387 - accuracy: 0.9624 - val_loss: 1.9777 - val_accuracy: 0.9573
Epoch 60/100
59/59 [=====] - 19s 314ms/step - loss: 1.8991 - accuracy: 0.9661 - val_loss: 1.9629 - val_accuracy: 0.9541
Epoch 61/100
59/59 [=====] - 19s 316ms/step - loss: 1.8711 - accuracy: 0.9634 - val_loss: 1.9111 - val_accuracy: 0.9584
Epoch 62/100
59/59 [=====] - 19s 318ms/step - loss: 1.8412 - accuracy: 0.9661 - val_loss: 1.8785 - val_accuracy: 0.9552
Epoch 63/100
59/59 [=====] - 19s 317ms/step - loss: 1.8098 - accuracy: 0.9672 - val_loss: 1.8545 - val_accuracy: 0.9605
Epoch 64/100
59/59 [=====] - 19s 315ms/step - loss: 1.7811 - accuracy: 0.9666 - val_loss: 1.8323 - val_accuracy: 0.9605
Epoch 65/100
59/59 [=====] - 19s 315ms/step - loss: 1.7539 - accuracy: 0.9658 - val_loss: 1.7975 - val_accuracy: 0.9573
Epoch 66/100
59/59 [=====] - 18s 312ms/step - loss: 1.7317 - accuracy: 0.9616 - val_loss: 1.7644 - val_accuracy: 0.9584
Epoch 67/100
59/59 [=====] - 19s 314ms/step - loss: 1.6939 - accuracy: 0.9693 - val_loss: 1.7387 - val_accuracy: 0.9584
Epoch 68/100
59/59 [=====] - 18s 312ms/step - loss: 1.6656 - accuracy: 0.9666 - val_loss: 1.7103 - val_accuracy: 0.9584
Epoch 69/100
59/59 [=====] - 18s 312ms/step - loss: 1.6420 - accuracy: 0.9706 - val_loss: 1.6789 - val_accuracy: 0.9573
Epoch 70/100
59/59 [=====] - 18s 311ms/step - loss: 1.6236 - accuracy: 0.9624 - val_loss: 1.6581 - val_accuracy: 0.9605
Epoch 71/100
59/59 [=====] - 18s 310ms/step - loss: 1.5939 - accuracy: 0.9664 - val_loss: 1.6489 - val_accuracy: 0.9584
Epoch 72/100
59/59 [=====] - 18s 311ms/step - loss: 1.5639 - accuracy: 0.9712 - val_loss: 1.6207 - val_accuracy: 0.9584
Epoch 73/100
59/59 [=====] - 18s 311ms/step - loss: 1.5568 - accuracy: 0.9573 - val_loss: 1.6466 - val_accuracy: 0.9445
Epoch 74/100
59/59 [=====] - 18s 313ms/step - loss: 1.5190 - accuracy: 0.9674 - val_loss: 1.5862 - val_accuracy: 0.9562
Epoch 75/100
59/59 [=====] - 18s 313ms/step - loss: 1.4930 - accuracy: 0.9685 - val_loss: 1.5712 - val_accuracy: 0.9541
Epoch 76/100
59/59 [=====] - 19s 314ms/step - loss: 1.4684 - accuracy: 0.9688 - val_loss: 1.5299 - val_accuracy: 0.9573
Epoch 77/100
59/59 [=====] - 18s 311ms/step - loss: 1.4418 - accuracy: 0.9720 - val_loss: 1.4877 - val_accuracy: 0.9530
Epoch 78/100
59/59 [=====] - 18s 312ms/step - loss: 1.4258 - accuracy: 0.9672 - val_loss: 1.4815 - val_accuracy: 0.9573
Epoch 79/100
59/59 [=====] - 18s 314ms/step - loss: 1.3998 - accuracy: 0.9712 - val_loss: 1.4486 - val_accuracy: 0.9594
Epoch 80/100
59/59 [=====] - 19s 314ms/step - loss: 1.3811 - accuracy: 0.9709 - val_loss: 1.4557 - val_accuracy: 0.9552
Epoch 81/100
59/59 [=====] - 18s 313ms/step - loss: 1.3627 - accuracy: 0.9661 - val_loss: 1.4170 - val_accuracy: 0.9573
Epoch 82/100
59/59 [=====] - 18s 313ms/step - loss: 1.3399 - accuracy: 0.9693 - val_loss: 1.3800 - val_accuracy: 0.9594
Epoch 83/100
59/59 [=====] - 18s 313ms/step - loss: 1.3127 - accuracy: 0.9714 - val_loss: 1.3665 - val_accuracy: 0.9594
Epoch 84/100
59/59 [=====] - 19s 315ms/step - loss: 1.2969 - accuracy: 0.9725 - val_loss: 1.3405 - val_accuracy: 0.9594
Epoch 85/100
59/59 [=====] - 18s 312ms/step - loss: 1.2756 - accuracy: 0.9714 - val_loss: 1.3296 - val_accuracy: 0.9584
Epoch 86/100
59/59 [=====] - 18s 313ms/step - loss: 1.2481 - accuracy: 0.9757 - val_loss: 1.3299 - val_accuracy: 0.9562
Epoch 87/100
59/59 [=====] - 18s 312ms/step - loss: 1.2443 - accuracy: 0.9698 - val_loss: 1.2867 - val_accuracy: 0.9605
Epoch 88/100
59/59 [=====] - 18s 313ms/step - loss: 1.2128 - accuracy: 0.9797 - val_loss: 1.2649 - val_accuracy: 0.9573
Epoch 89/100
59/59 [=====] - 18s 312ms/step - loss: 1.1986 - accuracy: 0.9741 - val_loss: 1.2515 - val_accuracy: 0.9616
Epoch 90/100
59/59 [=====] - 18s 312ms/step - loss: 1.1826 - accuracy: 0.9717 - val_loss: 1.2339 - val_accuracy: 0.9594
Epoch 91/100
59/59 [=====] - 19s 316ms/step - loss: 1.1693 - accuracy: 0.9709 - val_loss: 1.2162 - val_accuracy: 0.9605
Epoch 92/100
59/59 [=====] - 19s 314ms/step - loss: 1.1467 - accuracy: 0.9725 - val_loss: 1.2037 - val_accuracy: 0.9584
Epoch 93/100
59/59 [=====] - 18s 312ms/step - loss: 1.1270 - accuracy: 0.9725 - val_loss: 1.1811 - val_accuracy: 0.9594
Epoch 94/100
59/59 [=====] - 18s 311ms/step - loss: 1.1452 - accuracy: 0.9610 - val_loss: 1.2027 - val_accuracy: 0.9541
Epoch 95/100
59/59 [=====] - 19s 316ms/step - loss: 1.0980 - accuracy: 0.9738 - val_loss: 1.1526 - val_accuracy: 0.9584
Epoch 96/100

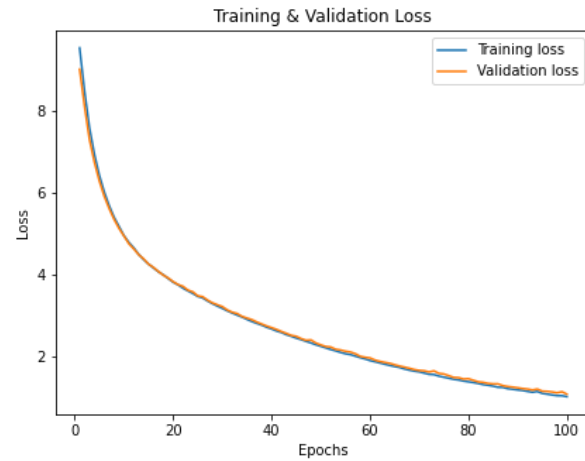
```

59/59 [=====] - 19s 314ms/step - loss: 1.0805 - accuracy: 0.9744 - val_loss: 1.1476 - val_accuracy: 0.9573
Epoch 97/100
59/59 [=====] - 18s 312ms/step - loss: 1.0600 - accuracy: 0.9797 - val_loss: 1.1339 - val_accuracy: 0.9594
Epoch 98/100
59/59 [=====] - 18s 313ms/step - loss: 1.0436 - accuracy: 0.9773 - val_loss: 1.1126 - val_accuracy: 0.9573
Epoch 99/100
59/59 [=====] - 18s 313ms/step - loss: 1.0413 - accuracy: 0.9744 - val_loss: 1.1382 - val_accuracy: 0.9413
Epoch 100/100
59/59 [=====] - 19s 314ms/step - loss: 1.0185 - accuracy: 0.9773 - val_loss: 1.0789 - val_accuracy: 0.9584

```

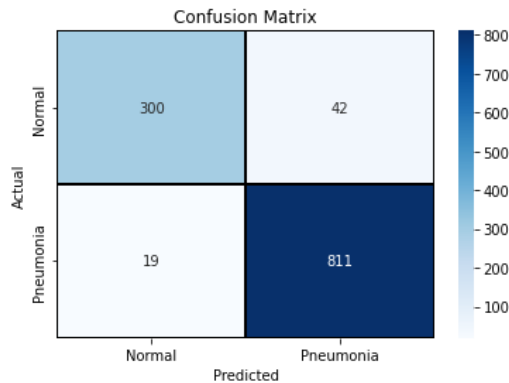
```
In [ ]: learn_curves(history7)
```

Final Training Accuracy: 0.9773151874542236
Final Validation Accuracy: 0.9583777785301208



```
In [ ]: evaluation(cnn7)
```

```
37/37 [=====] - 1s 17ms/step
```



```

37/37 [=====] - 1s 15ms/step - loss: 1.1008 - accuracy: 0.9480
[1.1008445024490356, 0.9479522109031677]

```

Evaluation

CNN7, though its validation and test accuracies were not the highest, minimized the number of false negatives and seemed to generalize better than the other models. Using a complex model to pick up as many features as possible, while using regularization to keep weights in check, seems to strike the right balance between generalization and error rate.

The biggest drawback is the preprocessing layer adding a not-so-insignificant chunk of computation time. Callbacks helped, but `min_delta` should probably be higher.

The model features two blocks of two stacks of conv2d layers with max pooling, a third such block with the first convolutional layer l2 regularized and dropout on the second at 50%.

All filters are doubled in number from previous iteration, and filter sizes remain the same.

```
In [ ]:
from keras.models import load_model

cnn7.save('my_model.h5') # creates a HDF5 file 'my_model.h5'
del cnn7 # deletes the existing model

# returns a compiled model identical to the previous one
model = load_model('my_model.h5')
```

```
In [ ]: final = models.load_model('my_model.h5')
```

```
In [ ]: #final training run
```

```
final.evaluate(X_train, y_train)
```

```
147/147 [=====] - 3s 16ms/step - loss: 1.0538 - accuracy: 0.9744
```

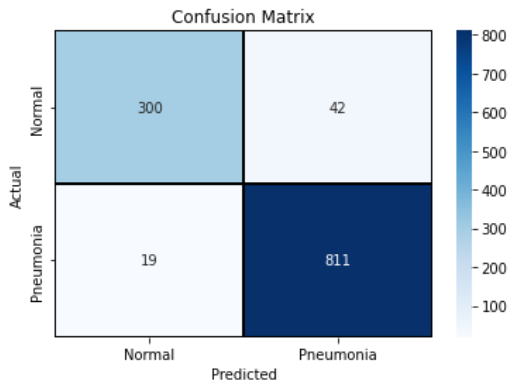
```
Out[ ]: [1.0538090467453003, 0.9743808507919312]
```

```
In [ ]:
```

```
# prediction and evaluation of Model 7
test_pred = final.predict((X_test<=0.49999)*1)
evaluation(final)
```

```
37/37 [=====] - 1s 20ms/step
```

```
37/37 [=====] - 1s 13ms/step
```



```
37/37 [=====] - 1s 15ms/step - loss: 1.1008 - accuracy: 0.9480
```

```
[1.1008445024490356, 0.9479522109031677]
```

Recommendations

1. Positive cases should be confirmed by trained medical personnel
2. Expand training with wider patient dataset (current Xrays were taken only from pediatric patients)
3. At almost 95% accuracy with 97% recall, the results can be relied upon to flag positive pneumonia cases.
4. Perfect for urgent care clinics and emergency departments where the potential exists for high volume and necessitates quick turn around.

```
In [ ]:
```