

Phase 4 Project: Detecting Pneumonia in Chest X-Rays using Convolutional Neural Networks

```
from tensorflow.random import set_seed
set_seed(321)
```

```
import numpy as np
np.random.seed(123)
```

```
!pip install lime
```

```
Collecting lime
```

```
  Downloading https://files.pythonhosted.org/packages/f5/86/91a13127d83d793e
```

```
    |████████████████████████████████████████| 276kB 12.1MB/s
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.6/
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.6/
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/c
Requirement already satisfied: cycloper>=0.10 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /us
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.6/c
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: pillow>=4.3.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/di
Building wheels for collected packages: lime
```

```
  Building wheel for lime (setup.py) ... done
```

```
  Created wheel for lime: filename=lime-0.2.0.1-cp36-none-any.whl size=283846
```

```
  Stored in directory: /root/.cache/pip/wheels/4c/4f/a5/0bc765457bd41378bf3ce8
```

```
Successfully built lime
```

```
Installing collected packages: lime
```

```
Successfully installed lime-0.2.0.1
```

```
# Import Statements
```

```
import os
```

```
import glob
```

```
from google.colab import drive
```

```
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,
```

```
from tensorflow.keras import models, layers, optimizers, regularizers, activations
```

```

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPl
from tensorflow.keras.applications import VGG16, VGG19

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils.class_weight import compute_class_weight

import lime
from lime import lime_base
from lime import lime_image
from skimage.segmentation import mark_boundaries

```

▼ Uploading the Data

```

# Mount Google Drive
drive.mount('/gdrive', force_remount=True)

Mounted at /gdrive

# Get current directory
print(os.getcwd())

# Get contents of the directory
print(os.listdir())

/content
['.config', 'drive', 'sample_data']

# Location of Zip File
drive_path = '/gdrive/MyDrive/Data/pneumonia_data.zip'
local_path = '/content'

!cp '{drive_path}' .

os.chdir(local_path)
!unzip -q 'pneumonia_data.zip'

```

▼ Data Preprocessing and Exploration

- Obtain and Store the Data in a train / test / val split
- Scale data
- Explore the data
 - check for class imbalance
 - look at images

```
# File Paths
```

```

path_train_pneumonia = '/content/chest_xray/train/PNEUMONIA'
path_train_normal = '/content/chest_xray/train/NORMAL'
path_test_pneumonia = '/content/chest_xray/test/PNEUMONIA'
path_test_normal = '/content/chest_xray/test/NORMAL'
path_val_pneumonia = '/content/chest_xray/val/PNEUMONIA'
path_val_normal = '/content/chest_xray/val/NORMAL'

```

```

# How many images in each set
print('Train Pneumonia', len(os.listdir(path_train_pneumonia)))
print('Train Normal', len(os.listdir(path_train_normal)))
print('Test Pneumonia', len(os.listdir(path_test_pneumonia)))
print('Test Normal', len(os.listdir(path_test_normal)))
print('Val Pneumonia', len(os.listdir(path_val_pneumonia)))
print('Val Normal', len(os.listdir(path_val_normal)))

```

```

Train Pneumonia 3875
Train Normal 1341
Test Pneumonia 390
Test Normal 234
Val Pneumonia 8
Val Normal 8

```

```

# Create data generators
# Ensure class_mode is binary
# Use batch size based on our above findings
# Use a target size of 224x224 px for each image
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    '/content/chest_xray/train',
    target_size=(224, 224),
    batch_size=5216,
    class_mode='binary',
    seed=123)

test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    '/content/chest_xray/test',
    target_size=(224, 224),
    batch_size=624,
    class_mode='binary',
    seed=123)

val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
    '/content/chest_xray/val',
    target_size=(224, 224),
    batch_size=16,
    class_mode='binary',
    seed=123)

```

```

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

```

```

# Create the data sets with the train/test/val splits
X_train, y_train = next(train_generator)
X_test, y_test = next(test_generator)
X_val, y_val = next(val_generator)

```

```

# Check our y_train and which classes are which

```

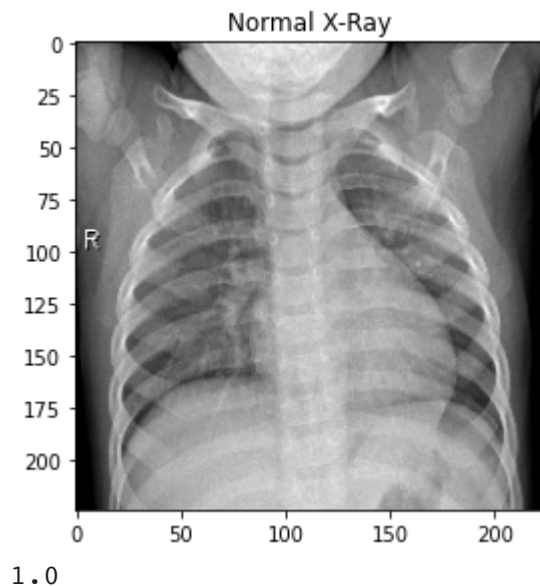
```
# Check out y_train and which classes are which
print(y_train[:20])
print(y_train.shape)
train_generator.class_indices

[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1.]
(5216,)
{'NORMAL': 0, 'PNEUMONIA': 1}
```

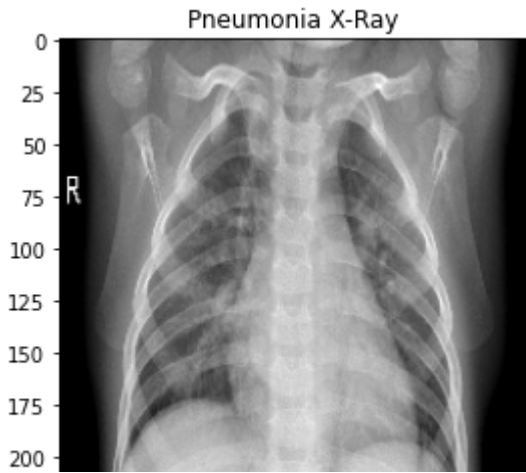
```
# Check shapes of the image data
print(X_train.shape)
print(X_test.shape)
print(X_val.shape)

(5216, 224, 224, 3)
(624, 224, 224, 3)
(16, 224, 224, 3)
```

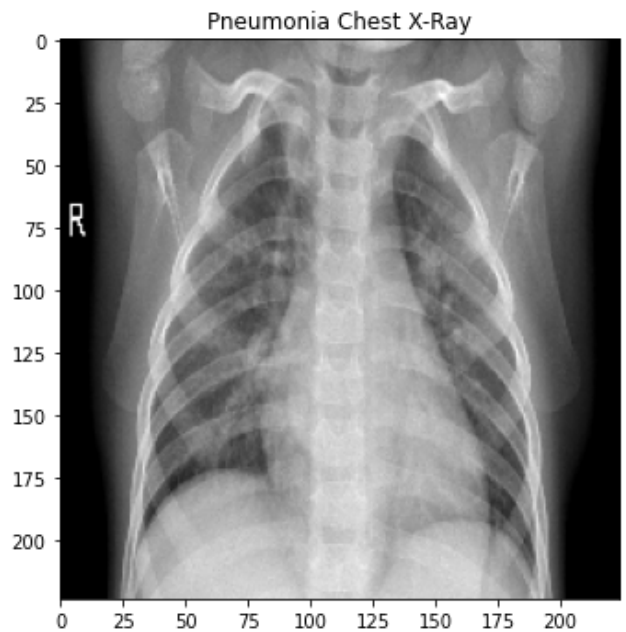
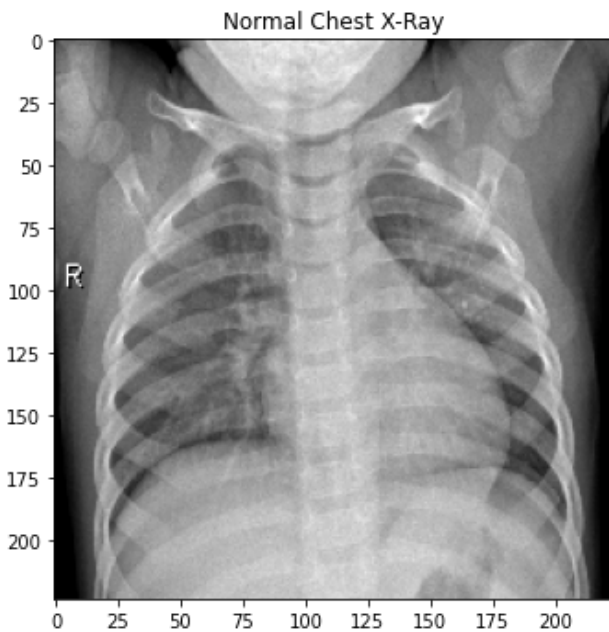
```
# What does a normal image look like?
plt.imshow(np.squeeze(X_train[0]))
plt.title('Normal X-Ray')
plt.tight_layout()
plt.show()
# Check the class label ('Normal':0)
print(y_train[8])
```



```
# What does a pneumonia xray look like?
plt.imshow(np.squeeze(X_train[1]))
plt.title('Pneumonia X-Ray')
plt.tight_layout()
plt.show()
# Check class label ('Pneumonia':1)
print(y_train[2])
```

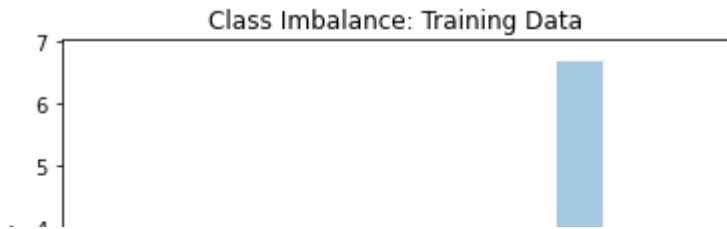


```
fig, axes = plt.subplots(1, 2, figsize=(12, 8))
axes[0].imshow(np.squeeze(X_train[0]))
axes[1].imshow(np.squeeze(X_train[1]))
axes[0].set_title('Normal Chest X-Ray')
axes[1].set_title('Pneumonia Chest X-Ray')
plt.show()
```



```
# Look at class imbalance of training data
sns.distplot(y_train)
plt.title('Class Imbalance: Training Data')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarning:
warnings.warn(msg, FutureWarning)
```



```
# Make a prettier visual
plt.figure(figsize=(10,7))
sns.barplot(x=[sum(y_train==0), sum(y_train==1)],
            y=['Normal', 'Pneumonia'],
            palette='Blues_d')
plt.title('Class Imbalance: Training Data')
plt.ylabel('Class')
plt.xlabel('Number of Training Images')
plt.show()
```



▼ Baseline CNN

- Baseline model with 3 convolutional layers, 3 max pooling layers, and one fully connected layer

```
baseline = models.Sequential()

baseline.add(Conv2D(32, (3, 3), activation='relu',
                    input_shape=(224, 224, 3)))
```

```

baseline.add(MaxPooling2D((2, 2)))

baseline.add(Conv2D(32, (3, 3), activation='relu'))
baseline.add(MaxPooling2D(2, 2))

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

baseline.add(Flatten())
baseline.add(Dense(64, activation='relu'))
baseline.add(Dense(1, activation='sigmoid'))

baseline.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['acc'])

# Look at summary to see all layers
baseline.summary()

```

Model: "sequential_23"

Layer (type)	Output Shape	Param #
conv2d_120 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_71 (MaxPooling)	(None, 111, 111, 32)	0
conv2d_121 (Conv2D)	(None, 109, 109, 32)	9248
max_pooling2d_72 (MaxPooling)	(None, 54, 54, 32)	0
conv2d_122 (Conv2D)	(None, 52, 52, 64)	18496
max_pooling2d_73 (MaxPooling)	(None, 26, 26, 64)	0
flatten_22 (Flatten)	(None, 43264)	0
dense_59 (Dense)	(None, 64)	2768960
dense_60 (Dense)	(None, 1)	65
Total params: 2,797,665		
Trainable params: 2,797,665		
Non-trainable params: 0		

```

history = baseline.fit(X_train,
                       y_train,
                       epochs=20,
                       batch_size=50,
                       validation_data=(X_val, y_val))

```

```

Epoch 1/20
105/105 [=====] - 3s 24ms/step - loss: 0.3547 - acc:
Epoch 2/20
105/105 [=====] - 2s 23ms/step - loss: 0.2216 - acc:
Epoch 3/20

```

```

105/105 [=====] - 2s 23ms/step - loss: 0.2217 - acc:
Epoch 4/20
105/105 [=====] - 2s 23ms/step - loss: 0.1969 - acc:
Epoch 5/20
105/105 [=====] - 2s 23ms/step - loss: 0.1773 - acc:
Epoch 6/20
105/105 [=====] - 2s 23ms/step - loss: 0.1596 - acc:
Epoch 7/20
105/105 [=====] - 2s 23ms/step - loss: 0.1504 - acc:
Epoch 8/20
105/105 [=====] - 2s 23ms/step - loss: 0.1687 - acc:
Epoch 9/20
105/105 [=====] - 2s 23ms/step - loss: 0.1367 - acc:
Epoch 10/20
105/105 [=====] - 2s 23ms/step - loss: 0.1260 - acc:
Epoch 11/20
105/105 [=====] - 2s 23ms/step - loss: 0.1144 - acc:
Epoch 12/20
105/105 [=====] - 2s 23ms/step - loss: 0.1054 - acc:
Epoch 13/20
105/105 [=====] - 2s 23ms/step - loss: 0.0980 - acc:
Epoch 14/20
105/105 [=====] - 2s 23ms/step - loss: 0.0898 - acc:
Epoch 15/20
105/105 [=====] - 2s 23ms/step - loss: 0.0891 - acc:
Epoch 16/20
105/105 [=====] - 2s 23ms/step - loss: 0.0782 - acc:
Epoch 17/20
105/105 [=====] - 2s 23ms/step - loss: 0.0777 - acc:
Epoch 18/20
105/105 [=====] - 2s 23ms/step - loss: 0.0723 - acc:
Epoch 19/20
105/105 [=====] - 2s 23ms/step - loss: 0.0663 - acc:
Epoch 20/20
105/105 [=====] - 2s 23ms/step - loss: 0.0597 - acc:

```

```

def evaluate_results(results):
    """
    input results of model fitting.
    output loss and accuracy curves, and confusion matrix
    """

    history = results.history
    plt.figure()
    plt.plot(history['loss'])
    plt.plot(history['val_loss'])
    plt.legend(['loss', 'val_loss'])
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.show()

    plt.figure()
    plt.plot(history['acc'])
    plt.plot(history['val_acc'])
    plt.legend(['acc', 'val_acc'])
    plt.title('Accuracy')
    plt.xlabel('Epochs')

```



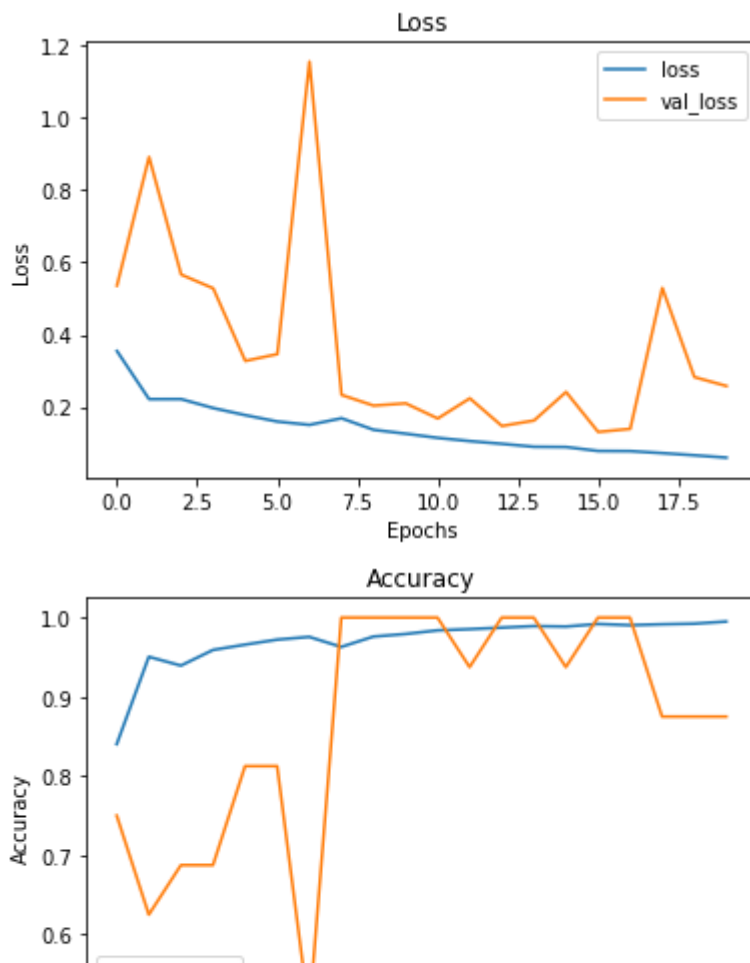
```
plt.ylabel('Accuracy')
plt.show()

y_hat_test = results.model.predict(X_test)
thresh = 0.5
y_pred = (y_hat_test < thresh).astype(np.int)
y_true = (y_test < thresh).astype(np.int)
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues_r', fmt='0.5g')
plt.xlabel('Predictions')
plt.ylabel('Actuals')
plt.title('Model Confusion Matrix')
plt.show()

print(classification_report(y_true, y_pred))
print('\n')

test_loss, test_acc = results.model.evaluate(X_test, y_test)
print('Test Loss, Acc:')
return test_loss, test_acc

evaluate_results(history)
```



Evaluation

This model is not great. It is mostly predicting that there is no pneumonia. This could be due to class imbalance. We can also see that the validation curves are very spiky before convergence, which could be due to the small validation set.

This model might be fixed with an early stopping callback. The model is also overfit quite a bit, so that would be another reason to include the early stopping, and possibly a learning rate reduction callback as well. I will use these changes in the model's second iteration.

Accuracy:

Iteration 2

- Address the class imbalance using class weights
 - based on the results of the baseline model's confusion matrix, which shows the model is predicting mostly one class
- Experiment with more epochs and early stopping
 - might prevent overfitting
- Reduce batch size
 - might improve accuracy

```
# Set up out class weights
```

```
weights = compute_class_weight('balanced', np.unique(y_train), y_train)
weights_dict = dict(zip(np.unique(y_train), weights))
weights_dict
```

```
{0.0: 1.9448173005219984, 1.0: 0.6730322580645162}
```

```
# Set up early stopping and learning rate reduction
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=5)
lr_redux = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5)
callbacks = [early_stop, lr_redux]
```

```
# Build model iteration 2 with improvements above and another conv layer
mod2 = models.Sequential()
```

```
mod2.add(Conv2D(32, (3, 3), activation='relu',
                input_shape=(224, 224, 3)))
mod2.add(MaxPooling2D((2, 2)))
```

```
mod2.add(Conv2D(32, (3, 3), activation='relu'))
mod2.add(MaxPooling2D(2, 2))
```

```
mod2.add(Conv2D(64, (3, 3), activation='relu'))
mod2.add(MaxPooling2D((2, 2)))
```

```
mod2.add(Flatten())
mod2.add(Dense(64, activation='relu'))
mod2.add(Dense(1, activation='sigmoid'))
```

```
mod2.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```

```
mod2.summary()
```

```
Model: "sequential_24"
```

Layer (type)	Output Shape	Param #
conv2d_123 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_74 (MaxPooling)	(None, 111, 111, 32)	0
conv2d_124 (Conv2D)	(None, 109, 109, 32)	9248
max_pooling2d_75 (MaxPooling)	(None, 54, 54, 32)	0
conv2d_125 (Conv2D)	(None, 52, 52, 64)	18496
max_pooling2d_76 (MaxPooling)	(None, 26, 26, 64)	0
flatten_23 (Flatten)	(None, 43264)	0
dense_61 (Dense)	(None, 64)	2768960
dense_62 (Dense)	(None, 1)	65

```

=====
Total params: 2,797,665
Trainable params: 2,797,665
Non-trainable params: 0
=====

```

```

results = mod2.fit(X_train,
                   y_train,
                   epochs=50,
                   batch_size=30,
                   validation_data=(X_val, y_val),
                   class_weight=weights_dict,
                   callbacks=callbacks)

```

```

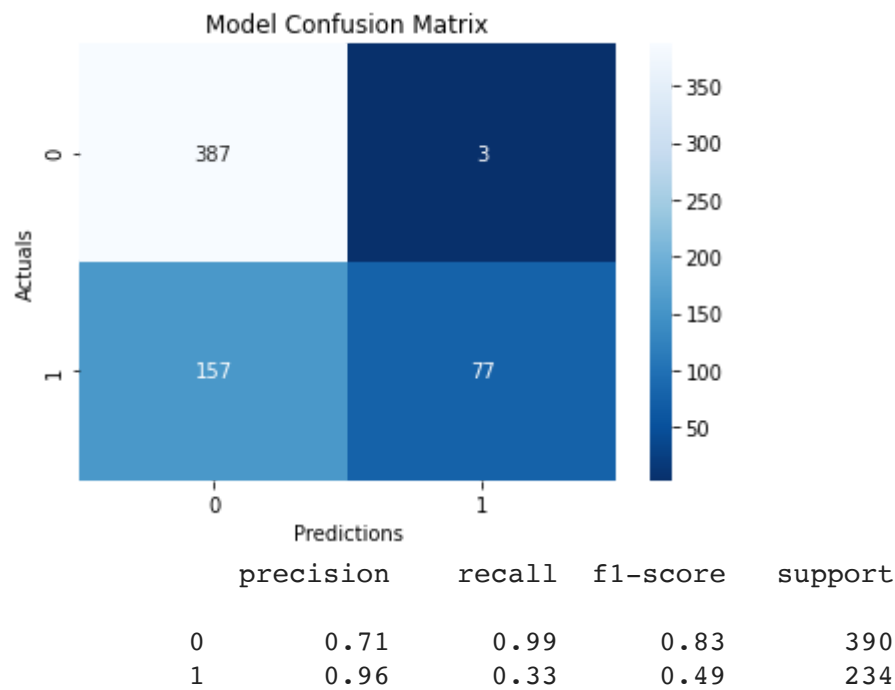
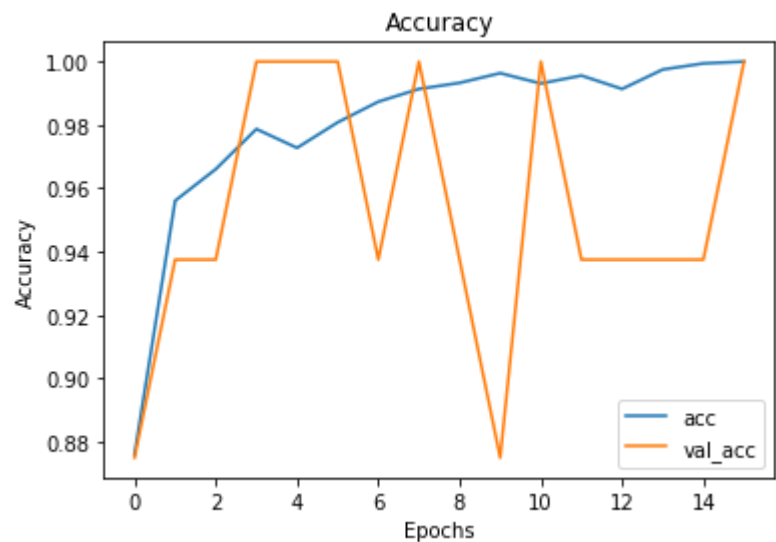
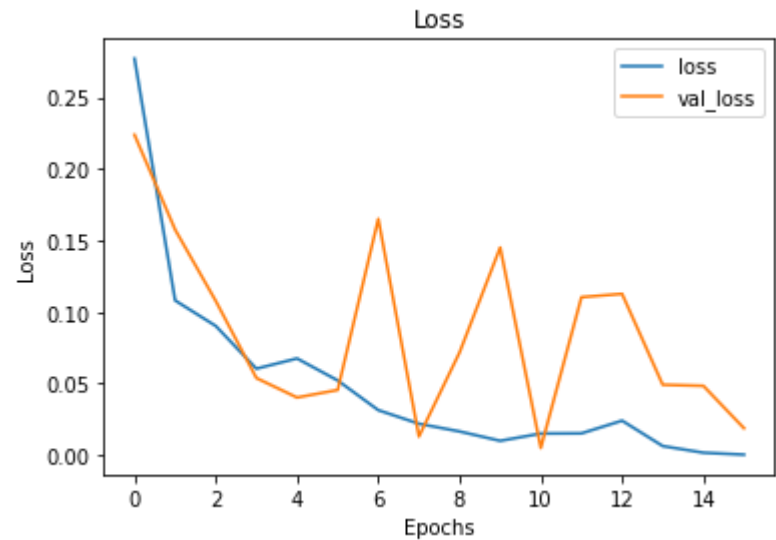
Epoch 1/50
174/174 [=====] - 3s 16ms/step - loss: 0.2772 - acc:
Epoch 2/50
174/174 [=====] - 3s 15ms/step - loss: 0.1080 - acc:
Epoch 3/50
174/174 [=====] - 3s 15ms/step - loss: 0.0902 - acc:
Epoch 4/50
174/174 [=====] - 3s 15ms/step - loss: 0.0603 - acc:
Epoch 5/50
174/174 [=====] - 3s 15ms/step - loss: 0.0675 - acc:
Epoch 6/50
174/174 [=====] - 3s 15ms/step - loss: 0.0521 - acc:
Epoch 7/50
174/174 [=====] - 3s 15ms/step - loss: 0.0313 - acc:
Epoch 8/50
174/174 [=====] - 3s 15ms/step - loss: 0.0218 - acc:
Epoch 9/50
174/174 [=====] - 3s 15ms/step - loss: 0.0164 - acc:
Epoch 10/50
174/174 [=====] - 3s 15ms/step - loss: 0.0099 - acc:
Epoch 11/50
174/174 [=====] - 3s 15ms/step - loss: 0.0150 - acc:
Epoch 12/50
174/174 [=====] - 3s 15ms/step - loss: 0.0151 - acc:
Epoch 13/50
174/174 [=====] - 3s 15ms/step - loss: 0.0240 - acc:
Epoch 14/50
174/174 [=====] - 3s 15ms/step - loss: 0.0062 - acc:
Epoch 15/50
174/174 [=====] - 3s 15ms/step - loss: 0.0015 - acc:
Epoch 16/50
174/174 [=====] - 3s 15ms/step - loss: 1.8764e-04 - a

```

```

evaluate_results(results)

```



Evaluation

We can see a slight improvement in this model from our baseline. Both the accuracy and loss improved a tiny bit.

There is still improvement to be made here. It is clear that the validation curves are not easily converging, which I believe might be due to their tiny size. The validation set here is only 16 images, while the training set is over 5000... Therefore, in the next iteration, I will try to rework that validation training split within the model fitting params instead of using the tiny validation

▼ Iteration 3

- Going off of iteration 2, I will try to rework the validation set size to see if it makes any improvements in the loss and accuracy curves.

```
mod3 = models.Sequential()

mod3.add(Conv2D(32, (3, 3), activation='relu',
               input_shape=(224, 224, 3)))
mod3.add(MaxPooling2D((2, 2)))

mod3.add(Conv2D(32, (3, 3), activation='relu'))
mod3.add(MaxPooling2D(2, 2))

mod3.add(Conv2D(64, (3, 3), activation='relu'))
mod3.add(MaxPooling2D((2, 2)))

mod3.add(Flatten())
mod3.add(Dense(64, activation='relu'))
mod3.add(Dense(1, activation='sigmoid'))

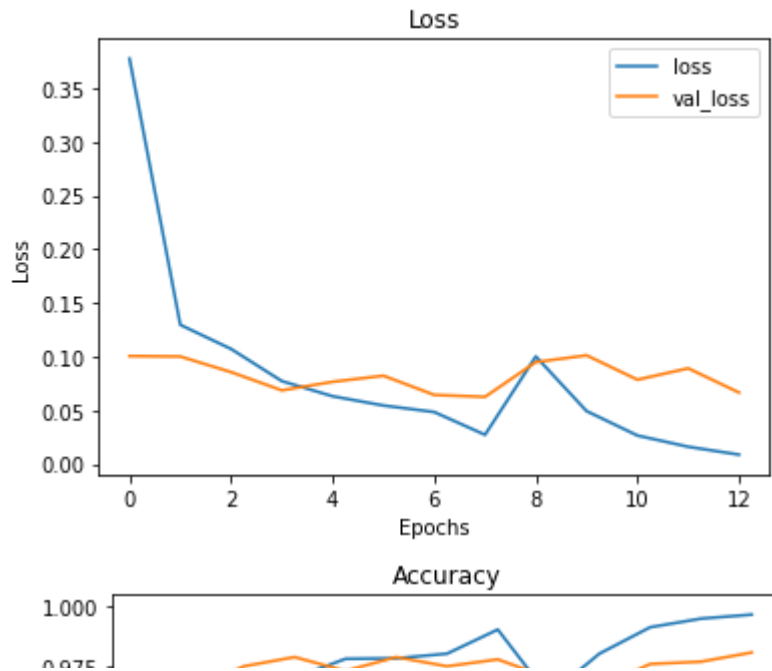
mod3.compile(loss='binary_crossentropy',
             optimizer='adam',
             metrics=['acc'])

results = mod3.fit(X_train,
                  y_train,
                  epochs=50,
                  batch_size=30,
                  validation_split=.2,
                  class_weight=weights_dict,
                  callbacks=callbacks)
```

```
Epoch 1/50
140/140 [=====] - 3s 18ms/step - loss: 0.3773 - acc:
Epoch 2/50
140/140 [=====] - 2s 17ms/step - loss: 0.1296 - acc:
Epoch 3/50
140/140 [=====] - 2s 17ms/step - loss: 0.1071 - acc:
Epoch 4/50
140/140 [=====] - 2s 17ms/step - loss: 0.0772 - acc:
Epoch 5/50
140/140 [=====] - 2s 17ms/step - loss: 0.0631 - acc:
Epoch 6/50
140/140 [=====] - 2s 17ms/step - loss: 0.0545 - acc:
Epoch 7/50
```

```
140/140 [=====] - 2s 17ms/step - loss: 0.0484 - acc:
Epoch 8/50
140/140 [=====] - 2s 17ms/step - loss: 0.0272 - acc:
Epoch 9/50
140/140 [=====] - 2s 17ms/step - loss: 0.1001 - acc:
Epoch 10/50
140/140 [=====] - 2s 17ms/step - loss: 0.0493 - acc:
Epoch 11/50
140/140 [=====] - 2s 17ms/step - loss: 0.0266 - acc:
Epoch 12/50
140/140 [=====] - 2s 17ms/step - loss: 0.0161 - acc:
Epoch 13/50
140/140 [=====] - 2s 17ms/step - loss: 0.0088 - acc:
```

```
evaluate_results(results)
```



Evaluation

The test set did not actually improve here but these curves look MUCH better. Therefore, in future iterations, I will keep this validation split instead of using the tiny validation set.

Iteration 4

- In this iteration, I want to experiment with a more complicated architecture
 - adding 2 convolutional layers per max pooling layer
 - when researching different successful image classification with xrays, this seemed to be a common practice
 - Since the previous models keep predicting mostly non-pneumonia, I think that adding more convolutional layers will help detect more of the edges/nuances in the images with pneumonia
- I will also add another 2 blocks of layers
 - AKA making the model deeper

```
mod4 = models.Sequential()

mod4.add(Conv2D(32, (3, 3), activation='relu',
                input_shape=(224, 224, 3)))
mod4.add(Conv2D(32, (3, 3), activation='relu'))
mod4.add(MaxPooling2D((2, 2)))

mod4.add(Conv2D(32, (3, 3), activation='relu'))
mod4.add(Conv2D(32, (3, 3), activation='relu'))
mod4.add(MaxPooling2D(2, 2))

mod4.add(Conv2D(64, (3, 3), activation='relu'))
mod4.add(Conv2D(64, (3, 3), activation='relu'))
```



```

mod4.add(MaxPooling2D((2, 2)))

mod4.add(Conv2D(64, (3, 3), activation='relu'))
mod4.add(Conv2D(64, (3, 3), activation='relu'))
mod4.add(MaxPooling2D((2, 2)))

mod4.add(Flatten())
mod4.add(Dense(128, activation='relu'))
mod4.add(Dense(64, activation='relu'))
mod4.add(Dense(1, activation='sigmoid'))

mod4.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['acc'])

mod4.summary()

```

Model: "sequential_29"

Layer (type)	Output Shape	Param #
conv2d_157 (Conv2D)	(None, 222, 222, 32)	896
conv2d_158 (Conv2D)	(None, 220, 220, 32)	9248
max_pooling2d_94 (MaxPooling)	(None, 110, 110, 32)	0
conv2d_159 (Conv2D)	(None, 108, 108, 32)	9248
conv2d_160 (Conv2D)	(None, 106, 106, 32)	9248
max_pooling2d_95 (MaxPooling)	(None, 53, 53, 32)	0
conv2d_161 (Conv2D)	(None, 51, 51, 64)	18496
conv2d_162 (Conv2D)	(None, 49, 49, 64)	36928
max_pooling2d_96 (MaxPooling)	(None, 24, 24, 64)	0
conv2d_163 (Conv2D)	(None, 22, 22, 64)	36928
conv2d_164 (Conv2D)	(None, 20, 20, 64)	36928
max_pooling2d_97 (MaxPooling)	(None, 10, 10, 64)	0
flatten_28 (Flatten)	(None, 6400)	0
dense_74 (Dense)	(None, 128)	819328
dense_75 (Dense)	(None, 64)	8256
dense_76 (Dense)	(None, 1)	65
Total params: 985,569		
Trainable params: 985,569		
Non-trainable params: 0		

```

results = mod4.fit(X_train,
                   y_train,
                   epochs=50,
                   batch_size=32,
                   validation_split=.2,
                   class_weight=weights_dict,
                   callbacks=[early_stop])

```

```

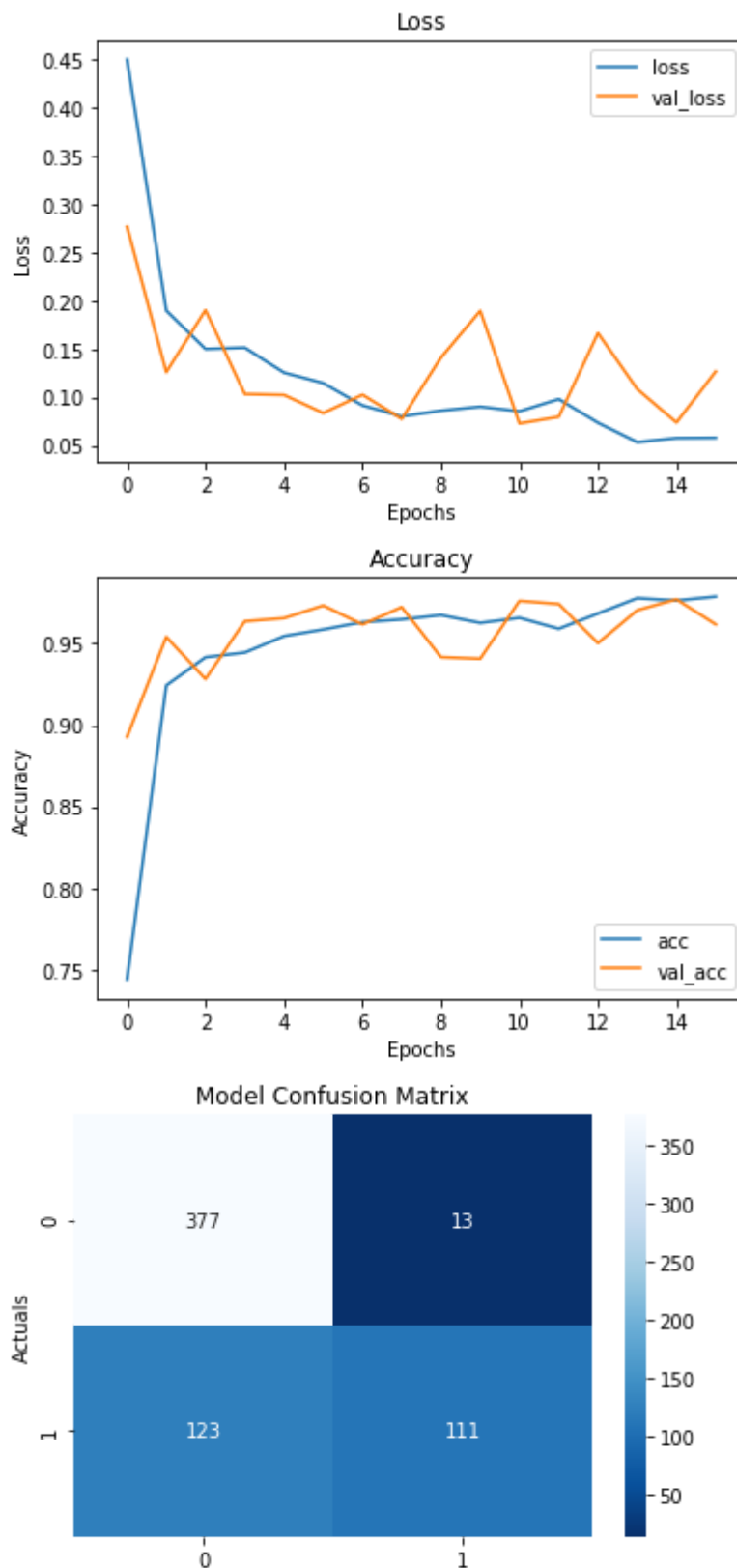
Epoch 1/50
131/131 [=====] - 4s 34ms/step - loss: 0.4504 - acc:
Epoch 2/50
131/131 [=====] - 4s 33ms/step - loss: 0.1898 - acc:
Epoch 3/50
131/131 [=====] - 4s 33ms/step - loss: 0.1499 - acc:
Epoch 4/50
131/131 [=====] - 4s 33ms/step - loss: 0.1512 - acc:
Epoch 5/50
131/131 [=====] - 4s 33ms/step - loss: 0.1254 - acc:
Epoch 6/50
131/131 [=====] - 4s 33ms/step - loss: 0.1145 - acc:
Epoch 7/50
131/131 [=====] - 4s 33ms/step - loss: 0.0913 - acc:
Epoch 8/50
131/131 [=====] - 4s 33ms/step - loss: 0.0800 - acc:
Epoch 9/50
131/131 [=====] - 4s 33ms/step - loss: 0.0859 - acc:
Epoch 10/50
131/131 [=====] - 4s 33ms/step - loss: 0.0899 - acc:
Epoch 11/50
131/131 [=====] - 4s 33ms/step - loss: 0.0852 - acc:
Epoch 12/50
131/131 [=====] - 4s 33ms/step - loss: 0.0979 - acc:
Epoch 13/50
131/131 [=====] - 4s 33ms/step - loss: 0.0734 - acc:
Epoch 14/50
131/131 [=====] - 4s 33ms/step - loss: 0.0532 - acc:
Epoch 15/50
131/131 [=====] - 4s 33ms/step - loss: 0.0574 - acc:
Epoch 16/50
131/131 [=====] - 4s 33ms/step - loss: 0.0577 - acc:

```

```

evaluate_results(results)

```



Evaluation

This did improve the model quite a bit. When we look at the classification report and the confusion matrix, we can see that this model is predicting the normal xrays very well. There are only a few test images that are normal that are misclassified by the model as having pneumonia (False Positives). On the other hand, it is still having trouble identifying the pneumonia photos. There are lots of False Negatives (images with pneumonia that are misclassified as normal). To try to fix this problem, I will continue to iterate on this model to improve the accuracy.

20/20 | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100%

▼ Iteration 5

- In this iteration, I will deepen the neural network to include more layers to hopefully pull more features out of the images to improve the model

```
mod5 = models.Sequential()

mod5.add(Conv2D(32, (3, 3), activation='relu',
               input_shape=(224, 224, 3)))
mod5.add(Conv2D(32, (3, 3), activation='relu'))
mod5.add(MaxPooling2D((2, 2)))

mod5.add(Conv2D(32, (3, 3), activation='relu'))
mod5.add(Conv2D(32, (3, 3), activation='relu'))
mod5.add(MaxPooling2D(2, 2))

mod5.add(Conv2D(64, (3, 3), activation='relu'))
mod5.add(Conv2D(64, (3, 3), activation='relu'))
mod5.add(MaxPooling2D((2, 2)))

mod5.add(Conv2D(64, (3, 3), activation='relu'))
mod5.add(Conv2D(64, (3, 3), activation='relu'))
mod5.add(MaxPooling2D((2, 2)))

mod5.add(Conv2D(128, (3, 3), activation='relu'))
mod5.add(Conv2D(128, (3, 3), activation='relu'))
mod5.add(MaxPooling2D((2, 2)))

mod5.add(Flatten())
mod5.add(Dense(128, activation='relu'))
mod5.add(Dense(64, activation='relu'))
mod5.add(Dense(1, activation='sigmoid'))

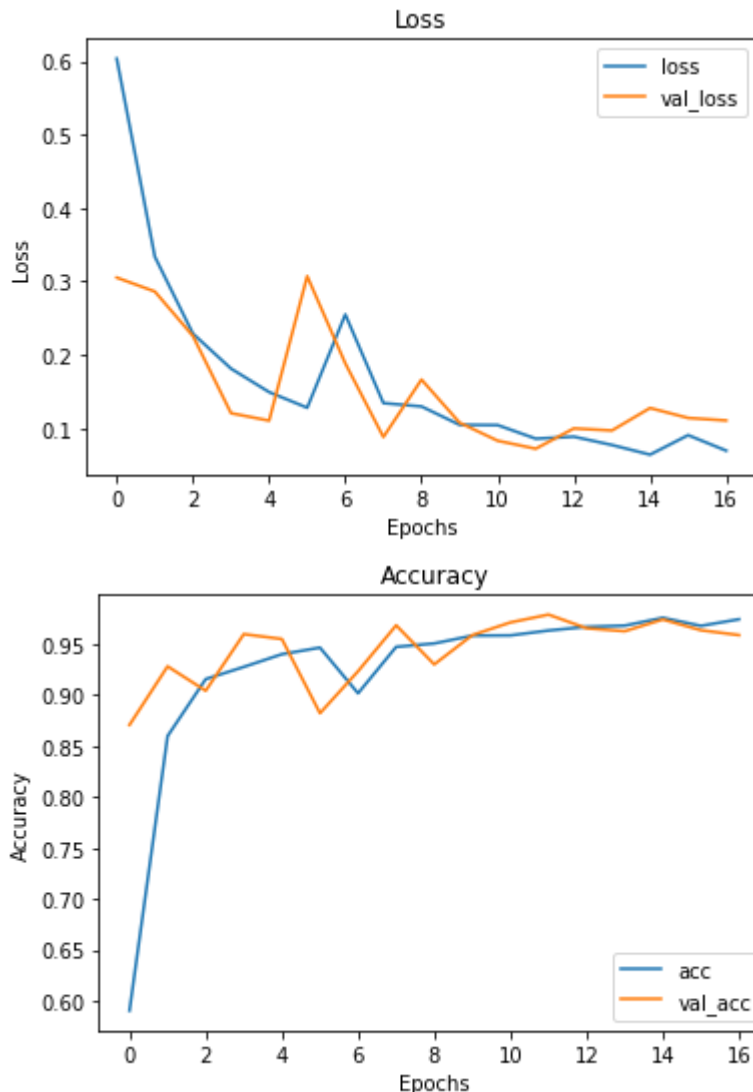
mod5.compile(loss='binary_crossentropy',
             optimizer='adam',
             metrics=['acc'])

results = mod5.fit(X_train,
                  y_train,
                  epochs=50,
                  batch_size=32,
                  validation_split=.2,
                  class_weight=weights_dict,
                  callbacks=[early_stop])

Epoch 1/50
131/131 [=====] - 5s 35ms/step - loss: 0.6035 - acc:
Epoch 2/50
131/131 [=====] - 4s 33ms/step - loss: 0.3336 - acc:
Epoch 3/50
131/131 [=====] - 4s 33ms/step - loss: 0.2287 - acc:
Epoch 4/50
131/131 [=====] - 4s 33ms/step - loss: 0.1810 - acc:
```

```
Epoch 5/50
131/131 [=====] - 4s 33ms/step - loss: 0.1491 - acc:
Epoch 6/50
131/131 [=====] - 4s 33ms/step - loss: 0.1278 - acc:
Epoch 7/50
131/131 [=====] - 4s 33ms/step - loss: 0.2548 - acc:
Epoch 8/50
131/131 [=====] - 4s 33ms/step - loss: 0.1341 - acc:
Epoch 9/50
131/131 [=====] - 4s 33ms/step - loss: 0.1295 - acc:
Epoch 10/50
131/131 [=====] - 4s 33ms/step - loss: 0.1045 - acc:
Epoch 11/50
131/131 [=====] - 4s 33ms/step - loss: 0.1041 - acc:
Epoch 12/50
131/131 [=====] - 4s 33ms/step - loss: 0.0856 - acc:
Epoch 13/50
131/131 [=====] - 4s 33ms/step - loss: 0.0883 - acc:
Epoch 14/50
131/131 [=====] - 4s 33ms/step - loss: 0.0770 - acc:
Epoch 15/50
131/131 [=====] - 4s 33ms/step - loss: 0.0638 - acc:
Epoch 16/50
131/131 [=====] - 4s 33ms/step - loss: 0.0904 - acc:
Epoch 17/50
131/131 [=====] - 4s 33ms/step - loss: 0.0693 - acc:
```

```
evaluate_results(results)
```



Evaluation

The accuracy is slowly improving here. We still have the issue of lots of False Negatives, but I will try to improve this with another strategy--Transfer Learning.

~ 250

Iteration 6 and 7: Using Transfer Learning

- The disadvantage of using deep learning is that sometime you don't have a big enough training set to make a very robust model. However, keras allows us to use models that have been pre-trained on millions of images as a base layer
 - In this iteration I will use the VGG19 model as my base and add my own layers on top to see if this improves my model's performance
- I will try 2 different methods

```
# Load our VGG19 model as our base
cnn_base = VGG19(weights='imagenet',
                  include_top=False,
                  input_shape=(224, 224, 3))
```

```

mod6 = models.Sequential()
mod6.add(cnn_base)
mod6.add(Flatten())
mod6.add(Dense(128, activation='relu'))
mod6.add(Dense(1, activation='sigmoid'))

```

```
cnn_base.summary()
```

Model: "vgg19"

Layer (type)	Output Shape	Param #
=====		
input_3 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
=====		
Total params: 20,024,384		
Trainable params: 20,024,384		
Non-trainable params: 0		

```
# Check which layers can be frozen
for layer in mod6.layers:
    print(layer.name, layer.trainable)
```

```
print(len(mod6.trainable_weights))
```

```
vgg19 True
flatten_30 True
dense_79 True
dense_80 True
36
```

```
# Freeze the model
cnn_base.trainable = False
```

```
# Double check that it has been frozen
for layer in mod6.layers:
    print(layer.name, layer.trainable)
```

```
print(len(mod6.trainable_weights))
```

```
vgg19 False
flatten_30 True
dense_79 True
dense_80 True
4
```

```
mod6.compile(loss='binary_crossentropy',
             optimizer='adam',
             metrics=['acc'])
```

```
mod6.summary()
```

```
Model: "sequential_31"
```

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 7, 7, 512)	20024384
flatten_30 (Flatten)	(None, 25088)	0
dense_79 (Dense)	(None, 128)	3211392
dense_80 (Dense)	(None, 1)	129
Total params: 23,235,905		
Trainable params: 3,211,521		
Non-trainable params: 20,024,384		

```
results = mod6.fit(X_train,
                  y_train,
                  epochs=50,
```



```

batch_size=32,
validation_split=.2,
class_weight=weights_dict,
callbacks=[early_stop])

```

```

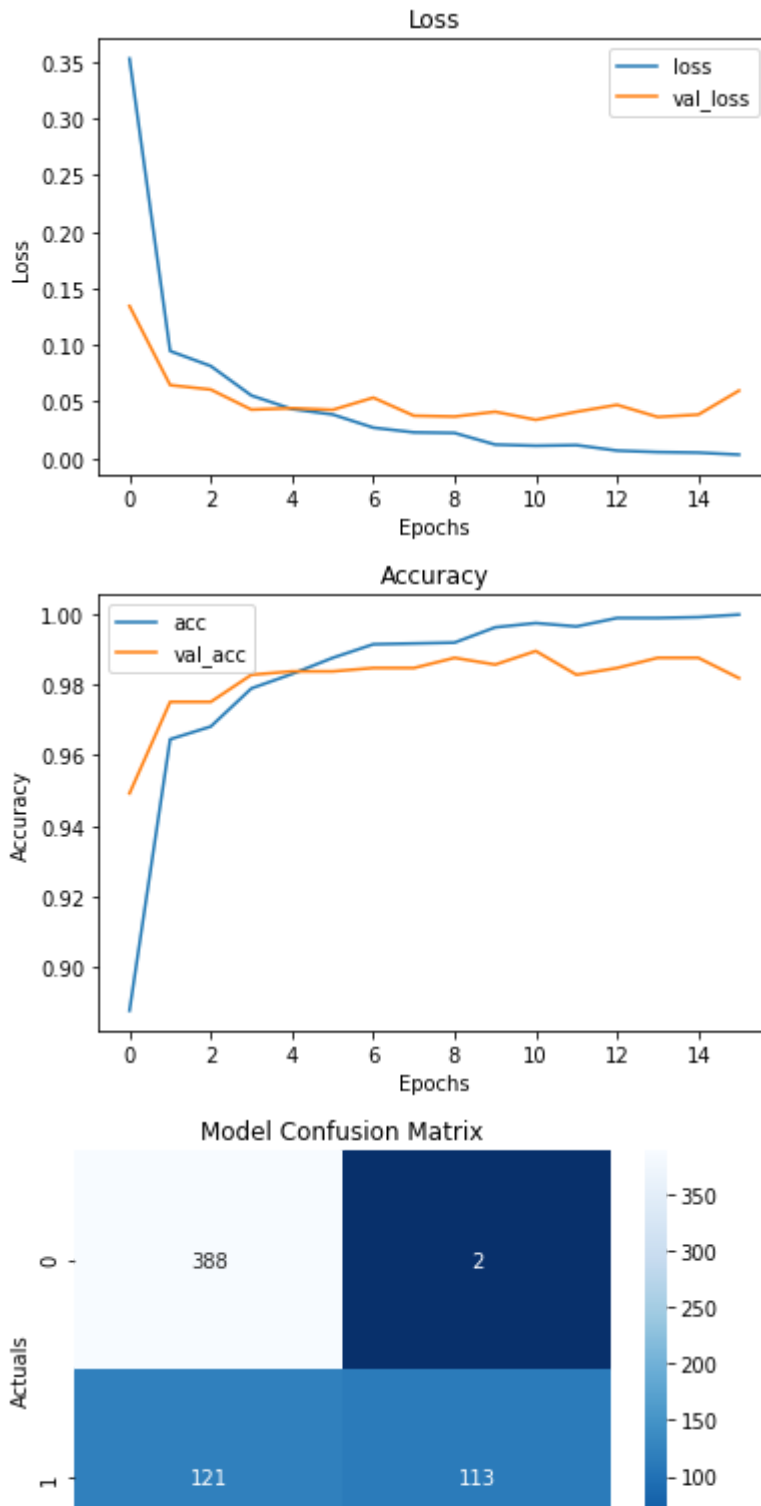
Epoch 1/50
131/131 [=====] - 8s 60ms/step - loss: 0.3528 - acc:
Epoch 2/50
131/131 [=====] - 8s 58ms/step - loss: 0.0947 - acc:
Epoch 3/50
131/131 [=====] - 8s 58ms/step - loss: 0.0815 - acc:
Epoch 4/50
131/131 [=====] - 8s 59ms/step - loss: 0.0555 - acc:
Epoch 5/50
131/131 [=====] - 8s 59ms/step - loss: 0.0435 - acc:
Epoch 6/50
131/131 [=====] - 8s 59ms/step - loss: 0.0387 - acc:
Epoch 7/50
131/131 [=====] - 8s 59ms/step - loss: 0.0271 - acc:
Epoch 8/50
131/131 [=====] - 8s 59ms/step - loss: 0.0229 - acc:
Epoch 9/50
131/131 [=====] - 8s 59ms/step - loss: 0.0224 - acc:
Epoch 10/50
131/131 [=====] - 8s 59ms/step - loss: 0.0122 - acc:
Epoch 11/50
131/131 [=====] - 8s 59ms/step - loss: 0.0112 - acc:
Epoch 12/50
131/131 [=====] - 8s 59ms/step - loss: 0.0117 - acc:
Epoch 13/50
131/131 [=====] - 8s 59ms/step - loss: 0.0068 - acc:
Epoch 14/50
131/131 [=====] - 8s 59ms/step - loss: 0.0056 - acc:
Epoch 15/50
131/131 [=====] - 8s 59ms/step - loss: 0.0051 - acc:
Epoch 16/50
131/131 [=====] - 8s 59ms/step - loss: 0.0033 - acc:

```

```

evaluate_results(results)

```



▼ Try another method

- `include_top = True`

```
mod7 = models.Sequential()
cnn_base2 = VGG19(weights='imagenet', include_top = True)
cnn_base2.trainable = False
mod7.add(cnn_base2)
mod7.add(Flatten())
mod7.add(Dense(64, activation='relu'))
mod7.add(Dense(1, activation='sigmoid'))
mod7.compile(optimizer='adam',
              loss='binary_crossentropy',
```

```
metrics=['acc'])
```

```
cnn_base2.summary()
```

```
Model: "vgg19"
```

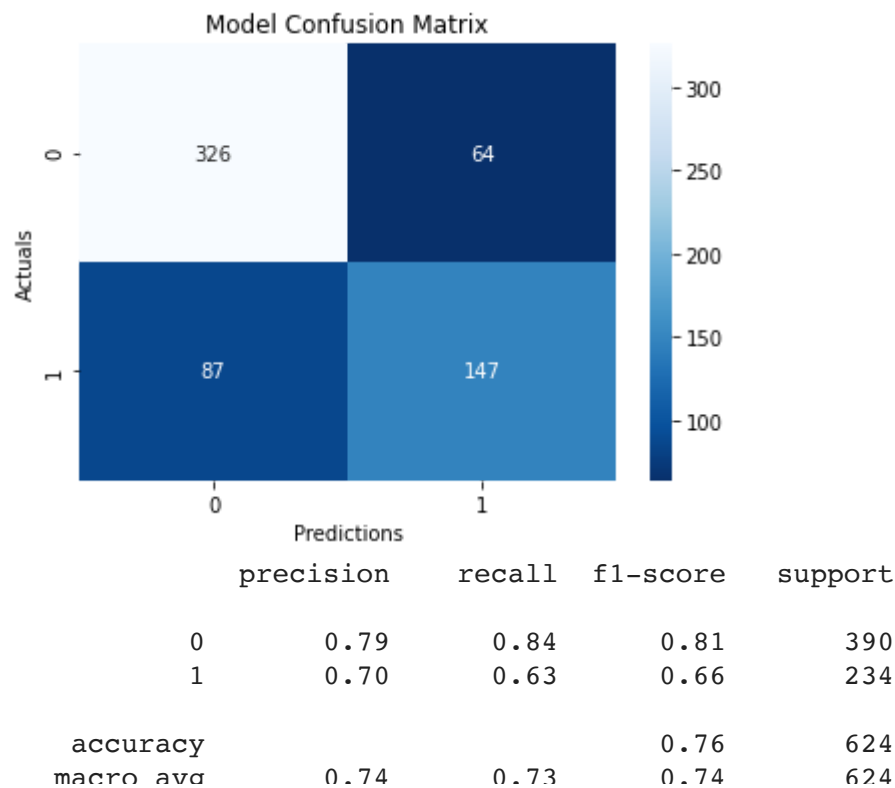
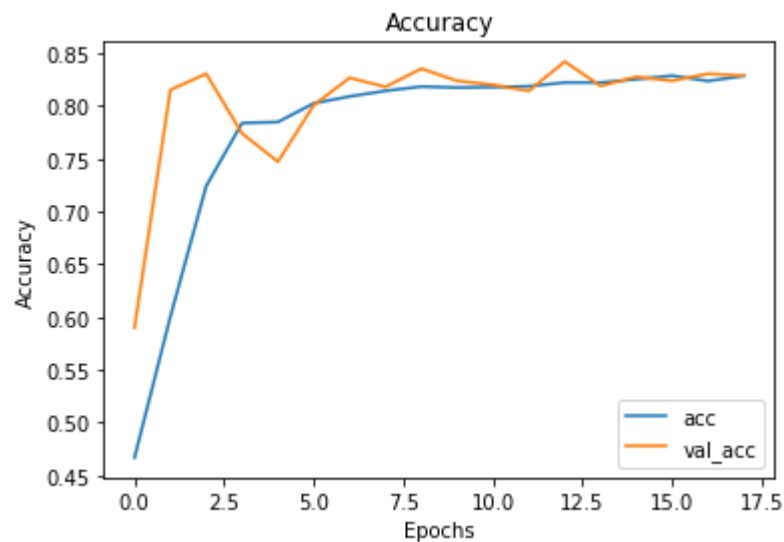
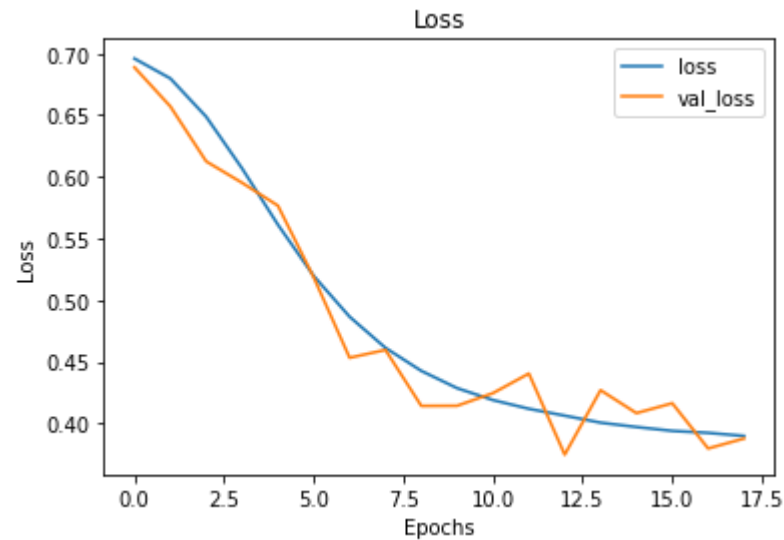
Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 143,667,240
 Trainable params: 0
 Non-trainable params: 143,667,240

```
results = mod7.fit(X_train,
                   y_train,
                   epochs=50,
                   batch_size=32,
                   validation_split=.2,
                   class_weight=weights_dict,
                   callbacks=[early_stop])
```

Epoch 1/50
 131/131 [=====] - 8s 61ms/step - loss: 0.6959 - acc:
 Epoch 2/50
 131/131 [=====] - 8s 59ms/step - loss: 0.6798 - acc:
 Epoch 3/50
 131/131 [=====] - 8s 60ms/step - loss: 0.6487 - acc:
 Epoch 4/50
 131/131 [=====] - 8s 60ms/step - loss: 0.6068 - acc:
 Epoch 5/50
 131/131 [=====] - 8s 60ms/step - loss: 0.5612 - acc:
 Epoch 6/50
 131/131 [=====] - 8s 60ms/step - loss: 0.5196 - acc:
 Epoch 7/50
 131/131 [=====] - 8s 60ms/step - loss: 0.4865 - acc:
 Epoch 8/50
 131/131 [=====] - 8s 60ms/step - loss: 0.4613 - acc:
 Epoch 9/50
 131/131 [=====] - 8s 59ms/step - loss: 0.4426 - acc:
 Epoch 10/50
 131/131 [=====] - 8s 60ms/step - loss: 0.4286 - acc:
 Epoch 11/50
 131/131 [=====] - 8s 60ms/step - loss: 0.4189 - acc:
 Epoch 12/50
 131/131 [=====] - 8s 59ms/step - loss: 0.4118 - acc:
 Epoch 13/50
 131/131 [=====] - 8s 60ms/step - loss: 0.4063 - acc:
 Epoch 14/50
 131/131 [=====] - 8s 60ms/step - loss: 0.4006 - acc:
 Epoch 15/50
 131/131 [=====] - 8s 60ms/step - loss: 0.3971 - acc:
 Epoch 16/50
 131/131 [=====] - 8s 60ms/step - loss: 0.3939 - acc:
 Epoch 17/50
 131/131 [=====] - 8s 60ms/step - loss: 0.3923 - acc:
 Epoch 18/50
 131/131 [=====] - 8s 60ms/step - loss: 0.3898 - acc:

```
evaluate_results(results)
```



Evaluation

Interestingly, these models did not perform better than the 5th iteration (without Transfer Learning). With more fine-tuning and unfreezing certain layers. However, the 7th iteration had a

much more balanced confusion matrix (more even False Positive and False Negatives) than past iterations.

▼ Which is the best model?

Defining 'Best' for this Use-Case:

In the case of diagnosing pneumonia, we want to reduce the number of False Negatives. Why? It is better to falsely label someone as having pneumonia (False Positive), than misclassifying someone who is actually sick who will then go untreated (False Negative).

Top 2 Models

When we look at all the models, model 5 has the best accuracy (over 80%), and less False Negatives than most of the other models. The second transfer learning model (model 7) has the least False Negatives, but more False Positives than model 5.

Final Selection

For this reason, I think the best model for this use-case, which reduces False Negatives but is still accurate, is model 5. Model 5 has an accuracy score of 82%, and the lowest number of False Negatives AND False Positives. Therefore, it is an overall better model than any of the previous iterations.

```
# Save Model 5 as final model
final_cnn_path = '/content/drive/MyDrive/final_pre_trained_cnn.hd5'

mod5.save(final_cnn_path)

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied aut
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied aut
INFO:tensorflow:Assets written to: /content/drive/MyDrive/final_pre_trained_cr

# Reload model
final_model = models.load_model(final_cnn_path)
```

▼ Look at the Model Explanations

- Look at the features of the image that are contributing to its class prediction

```
# Get random image and label
label = y_train[0]
img = X_train[0]
```

```
# Get model pred
```

```
# Get model pred
pred = mod5.predict(np.array([img]))
pred_class = int(pred.round())

# Print true class, predicted class and image
print('True Class:', label)
print('Predicted Class:', pred_class)
array_to_img(img)
```

```
True Class: 0.0
Predicted Class: 0
```



```
# Make an explainer
explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(img, mod5.predict, top_labels=2,
                                       hide_color=None, num_samples=2000)
```

100%

2000/2000 [02:58<00:00, 11.20it/s]

```
# Display features contributing to the pneumonia classification
temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                           positive_only=True,
                                           num_features=5)

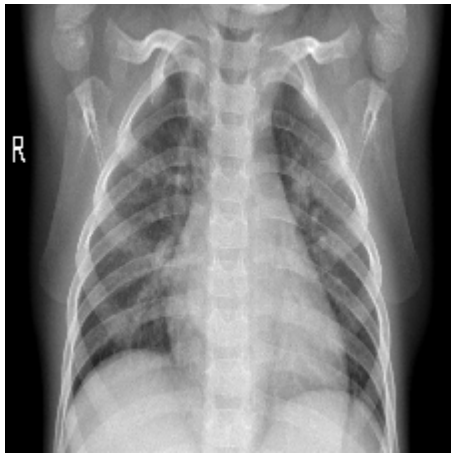
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
plt.title('Features Contributing to Normal Classification')
plt.show()
```

```
# Do the same with image at index 2
label2 = y_train[1]
img2 = X_train[1]

# Get model pred
pred2 = mod5.predict(np.array([img2]))
pred_class2 = int(pred.round())

# Print true class, predicted class and image
print('True Class:', label2)
print('Predicted Class:', pred_class2)
array_to_img(img2)
```

```
True Class: 1.0
Predicted Class: 0
```



```
explanation2 = explainer.explain_instance(img2, mod5.predict, top_labels=2,
                                         hide_color=None, num_samples=2000)

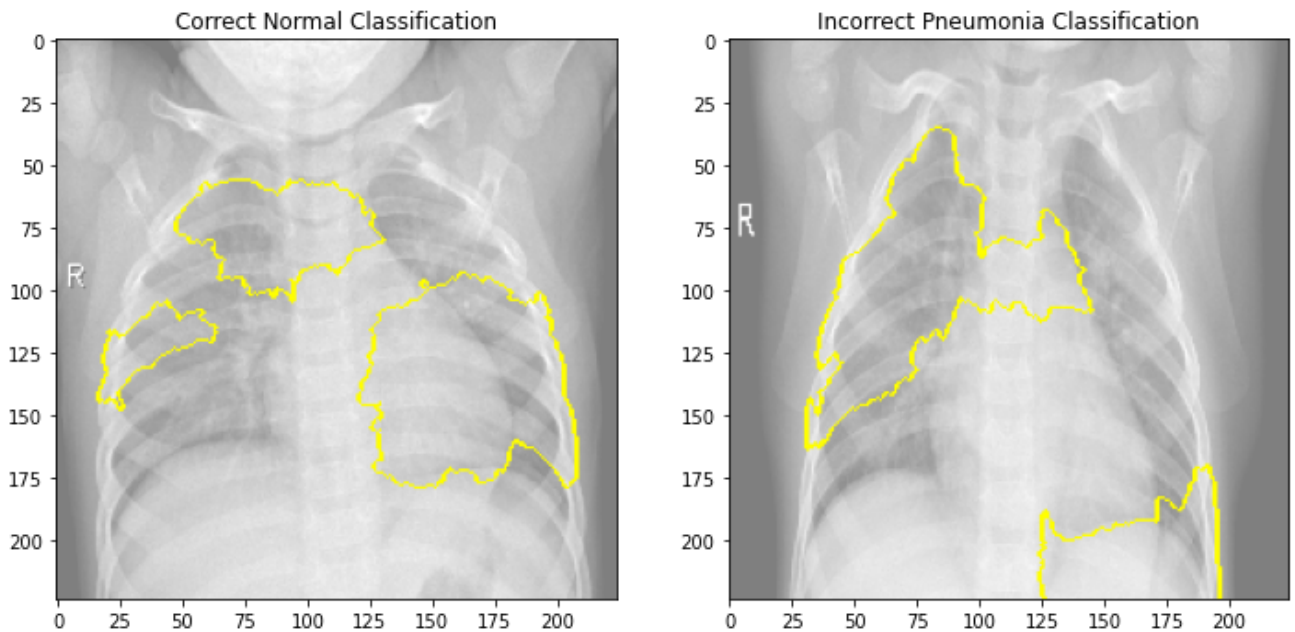
temp2, mask2 = explanation2.get_image_and_mask(explanation.top_labels[0],
                                                positive_only=True,
                                                num_features=5)

plt.imshow(mark_boundaries(temp2 / 2 + 0.5, mask2))
plt.title('Features Contributing to Pneumonia Classification')
plt.show()
```


100%

2000/2000 [00:13<00:00, 148.43it/s]

```
fig, axes = plt.subplots(1, 2, figsize=(12, 8))
axes[0].imshow(mark_boundaries(temp / 2 + 0.5, mask))
axes[1].imshow(mark_boundaries(temp2 / 2 + 0.5, mask2))
axes[0].set_title('Correct Normal Classification')
axes[1].set_title('Incorrect Pneumonia Classification')
plt.show()
```



Recommendations and Future Work

I would recommend to use this model as a tool for efficiency. Implementing this model in a radiology setting as way to assist x-ray technicians in detecting pneumonia would be the best way to use this technology. For example, once the chest x-ray is taken, it can automatically give its prediction to the tech. The tech and/or doctor would then need less time to review the model's prediction and use their trained eye for a final diagnosis. This would speed up the efficiency of the entire department so that the doctors' and techs' time can be mostly spent on other tasks.

Since this model has a lot of room for improvement, some future work could include:

- try to retrain the model using smaller images (ex. 64x64 px or 32x32 px)
- look at more preprocessing methods to remove the diaphragm and thus remove noise
- freeze and unfreeze specific layers of the transfer learning models to fine-tune them
- do more research on other methods to tune cnns specifically with chest x-ray/pneumonia classification