1 Phase 4 Project

Author: Freddy Abrahamson
Date created: 7-1-2022
Discipline: Data Science

1.1 Overview

1.1.1 In order for the notebook to run successfully:

1. Download the data from https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia)

2. The download should be unzipped to the same folder as where your Jupyter notebook is located. The unzipped folder is called 'chest_xray'.

1.1.2 Business Problem

In the United States about 1.5 million people were diagnosed with pneumonia in an emergency room, in 2018, and around 40,000 people died of pneumonia in that same year. Pneumonia is the world’s leading cause of death among children under 5 years of age, killing approximately 2,400 children a day in 2015. These facts underlie the importance, of not only a correct diagnosis, but a timely one. Now, with the introduction of COVID-19, and it's role as a precursor to pneumonia, the need for a cost effective, non-invasive diagnostic tool is even greater. A model that could correctly identify an x-ray as one of someone with pneumonia, would serve as a cost effective non-invasive tool, that could be used to prioritize which patients the doctor should see first.

https://www.cdc.gov/dotw/pneumonia/index.html (https://www.cdc.gov/dotw/pneumonia/index.html)
https://www.thoracic.org/patients/patient-resources/resources/top-pneumonia-facts.pdf (https://www.thoracic.org/patients/patient-resources/resources/top-pneumonia-facts.pdf)

1.1.3 Data Understanding and Data preparation

The data was taken from Kaggle.com. There are a total of 5856 images. This includes 1583 'normal' images, and 4273 'pneumonia' images. The ratio of 'pneumonia' images to 'normal' images is about 2.7 : 1. I divided all these images between train , test, and val folders at a ratio of .8:.1:.1 respectively. I maintained the 2.7 to 1 ratio between the 'pneumonia' images and 'normal' images, for all the folders. Once how many of each image would go to each folder was established, all the 'normal', and 'pneumonia' images were chosen randomly. The primary concern with the dataset preparation would be to normalize the image values. All the values were scaled to a range between 0 and 1.

1.1.4 Modeling

I used Keras and Tensorflow to create the models. Given that with the use of the filters, cnn(s) excel at detecting features of various sizes,I chose to use the less apt multi-layer perceptron as a baseline model. I then tried to overfit on purpose using a cnn. I began with a cnn model that has 4 activation layers for the feature extraction part,with the number of nodes for each each layer being 16,32,64, and 128 respectively. I used ReLu as my activation function for all feature detection, as well as for the classification layers. Given that this is a binary classification problem (0 for normal, and 1 for pneumonia), I used a sigmoid function for the output layer. From there, based on the results, I would either try to reduce the bias, by adding a layer, adding more nodes to existing layers, or both; or reduce the variance by increasing the filter size to improve generalizability, or add dropout layers.

1.1.5 Evaluation

Given the importance of correctly identifying a patient with pneumonia, my primary goal was to find a model that produced the best recall scores. To this end, I was looking for a model that would produce the best bias/variance combination between the train and test data sets. I did this by creating a function best_model(), which utilizes the auc() function from sklearn.metrics. The x-axis is represented by the absolute difference between the train and test scores, while the y-axis is represented by the test scores. The higher the test score, and the lower the train-test difference, the greater the area under the curve. The function returns a dataframe with the models, and their respective test scores, sorted by their auc. The model with the highest auc is the best. The secondary goal was a model that would have a good accuracy score, which the 'best' model in fact does, with a score over 90%.

2 Import Modules

In [1]:
```
import os
import time
import shutil
import random
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow.keras.metrics
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import confusion_matrix
from keras import models, layers, optimizers
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
%matplotlib inline
```

Creating random seeds for reproducibility.

In [2]:
```
# set random seeds
seed = 42
np.random.seed(seed)
tf.random.set_seed(seed)
```

3 Define Functions

3.1 best_model()

The best_model function returns the best train test score combination based on the auc function, where the difference between the test and the train scores represents the x axis, and test score represents the y axis. In order use the auc function, for each x,y coordinate we created a list of length three, with 0 and 1 at the ends, and the actual x,y values in the middle. The model with the highest auc score is the best.

This function takes as as an argument an integer which represents the total number of models.

This function returns a dataframe with five columns:

1. The model name
2. The recall score for the train set
3. The recall score for the test set
4. The absolute value of the difference between the two scores
5. The auc score, sorted in ascending order

In [3]:
```python
def best_model(n):
    from sklearn.metrics import auc
    scores_df = neural_network_model_scores_df(n)

#   creating 'test_scores' and 'score_diffs' zero populated lists of shape(rows,3)
    rows = len(scores_df)
    test_scores = np.zeros((rows, 3))
    score_diffs = np.zeros((rows, 3))
    auc_scores = []

#   populating 'test_scores' and 'score_diffs' so each list has a format [0,test_score,1],
#   and [0,score_diff,1] respectively
    for row in range(rows):
        test_scores[row][1] = scores_df['test score'][row]
        test_scores[row][2] = 1
        score_diffs[row][1] = scores_df['train-test diff'][row]
        score_diffs[row][2] = 1

#   creating a list of all the auc scores
    for row in range(rows):
        auc_score = auc(score_diffs[row], test_scores[row])
        auc_scores.append(auc_score)

#   getting the greatest auc score, and the index number of that row
    best_auc_score = max(auc_scores)
    best_score_index = auc_scores.index(best_auc_score)

#   add an auc_score cloumn to scores_df
    scores_df['auc score'] = auc_scores


#   return scores_df sprted by auc score
    return scores_df.sort_values(by='auc score', ascending=False)
```

3.2 neural_network_model_scores_df()

This function takes as as an argument an integer which represents the total number of models.

This function returns a dataframe with four columns:

1. The model name
2. The recall score for the train set
3. The recall score for the test set
4. The absolute value of the difference between the two scores

In [4]:
```
def neural_network_model_scores_df (n):
    count = 0
    model_dict ={}
    model_dict_list =[]
    while count<n:
        if count == 0:
            train_score = globals()['baseline_train_eval_dict']['recall']
            test_score = globals()['baseline_test_eval_dict']['recall']
            model_name = 'baseline_model'
            model_dict = {'model name':model_name, 'train score':train_score, 'test score':test_score}
            model_dict_list.append(model_dict)
            count+=1
        else:
            train_score = globals()['cnn_' +str(count) +'_train_eval_dict']['recall']
            test_score = globals()['cnn_' +str(count) +'_test_eval_dict']['recall']
            model_name = 'cnn_model_'+ str(count)
            model_dict = {'model name':model_name, 'train score':train_score, 'test score':test_score}
            model_dict_list.append(model_dict)
            count+=1
    scores_df = pd.DataFrame(model_dict_list)
    scores_df['train-test diff'] = abs(scores_df['train score'] - scores_df['test score'])
    return scores_df
```

3.3 recall_dict()

This function takes as an as an argument a dictionary returned by the evaluate() method from a tensorflow model, that includes recall, among it's metrics.

This function returns the following:

1. The key/value pair corresponding to the recall score

In [5]:
```
def recall_dict(eval_dict):
    eval_dict = {key: eval_dict[key] for key in eval_dict.keys()
                 & {'recall'}}
    for key in eval_dict.keys():
        eval_dict[key] = round(eval_dict[key],4)
    return eval_dict
```

3.4 plot_confusion_matrix()

Note: The code for this function was taken from the instructional website: https://deeplizard.com/learn/video/km7pxKy4UHU (https://deeplizard.com/learn/video/km7pxKy4UHU)

This function takes the following arguments:

1. cm: The array representing the results from the scikit-learn confusion_matrix() function
2. classes: an array with the name of the class labels of the confusion matrix.
3. normalize: if True, normalizes the values displayed by the confusion matrix. Default is False.
4. title: The title of the confusion matrix. Default is 'Confusion Matrix'.
5. cmap: Matplotlib colormap. Default is plt.cm.Blues.

This function returns a graphical plot of the confusion matrix.

In [6]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')


    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

3.5 visualize_training_results()

This function takes as an as an argument the variable returned by fitting a tensorflow model. it is of type: tensorflow.python.keras.callbacks.History

The best_model function returns the following plot:

1. The change in recall(y-axis) with respect to the number of epochs(x-axis)for both the train and validation sets

In [7]:
```
def visualize_training_results(results):
    history = results.history
    plt.figure()
    plt.plot(history['recall'])
    plt.plot(history['val_recall'])
    plt.legend(['recall', 'val_recall'])
    plt.title('Train Recall and Validation Recall')
    plt.xlabel('Epochs')
    plt.ylabel('Train Recall and Val Recall')
    plt.show()
```

4 Importing and Organizing Data

I will create two new folders: 'all_normal' and 'all_pneumonia', and copy all the corresponding images from the 'chest_xray' folder to these folders. I will then create a folder called 'train_test_val', with three folders inside of it:'train', 'test', and 'val'. Each of these three folders will have a 'normal', and a 'pneumonia' folder. I will randomly copy the images from the 'all_normal', and 'all_pneumonia' folders, to these three folders, with a split of 80%,10%,10% respectively, keeping the ratio between the number of 'normal' and 'pneumonia' images uniform across all three folders (stratified). The ratio of pneumonia images to normal images is 2.7:1.

In [8]:
```
# creating 'all_normal' and 'all_pneumonia' folders
all_normal_dir = 'all_normal/'
all_pneumonia_dir = 'all_pneumonia/'
os.mkdir(all_normal_dir)
os.mkdir(all_pneumonia_dir)
```

Creating the 'train_test_val' folder and all of its subfolders

In [9]:
```
# creating train_test_val folders:

# train_test_val parent folder:
train_test_val_dir = 'train_test_val/'
os.mkdir(train_test_val_dir)

# train folders:
train_dir = 'train_test_val/train'
train_normal_dir = 'train_test_val/train/normal'
train_pneumonia_dir = 'train_test_val/train/pneumonia'
os.mkdir(train_dir)
os.mkdir(train_normal_dir)
os.mkdir(train_pneumonia_dir)

# test folders:
test_dir = 'train_test_val/test'
test_normal_dir = 'train_test_val/test/normal'
test_pneumonia_dir = 'train_test_val/test/pneumonia'
os.mkdir(test_dir)
os.mkdir(test_normal_dir)
os.mkdir(test_pneumonia_dir)

# val folders:
val_dir = 'train_test_val/val'
val_normal_dir = 'train_test_val/val/normal'
val_pneumonia_dir = 'train_test_val/val/pneumonia'
os.mkdir(val_dir)
os.mkdir(val_normal_dir)
os.mkdir(val_pneumonia_dir)
```

Creating lists with name of the files that are in the 'pneumonia' and 'normal' folders contained in the 'chest_xray'folder. There are there sets of 'pneumonia' and 'normal' folders. One for each of the train, test and val folders. There are 6 lists total.

In [10]:
```
# creating a list with the file names for the respective original 'NORMAL' folders
normal_train_imgs = [file for file in os.listdir('chest_xray/train/NORMAL') if file.endswith('.jpeg')]
normal_test_imgs = [file for file in os.listdir('chest_xray/test/NORMAL') if file.endswith('.jpeg')]
normal_val_imgs = [file for file in os.listdir('chest_xray/val/NORMAL') if file.endswith('.jpeg')]

# creating a list with the file names for the respective original 'PNEUMONIA' folders
pneumonia_train_imgs = [file for file in os.listdir('chest_xray/train/PNEUMONIA') if file.endswith('.jpe
pneumonia_test_imgs = [file for file in os.listdir('chest_xray/test/PNEUMONIA') if file.endswith('.jpeg'
pneumonia_val_imgs = [file for file in os.listdir('chest_xray/val/PNEUMONIA') if file.endswith('.jpeg')
```

I will use the lists to copy all the files to their corresponding folder, either 'all_normal' or 'all_pneumonia'.

In [11]:
```python
# copying all the images from the original normal train folder to 'all_normal' folder
for img in normal_train_imgs:
    origin = os.path.join('chest_xray/train/NORMAL', img)
    destination = os.path.join('all_normal/', img)
    shutil.copyfile(origin, destination)

# copying all the images from the original normal test folder to 'all_normal' folder
for img in normal_test_imgs:
    origin = os.path.join('chest_xray/test/NORMAL', img)
    destination = os.path.join('all_normal/', img)
    shutil.copyfile(origin, destination)

# copying all the images from the original normal val folder to 'all_normal' folder
for img in normal_val_imgs:
    origin = os.path.join('chest_xray/val/NORMAL', img)
    destination = os.path.join('all_normal/', img)
    shutil.copyfile(origin, destination)

# copying all the images from the original pneumonia train folder to 'all_pneumonia' folder
for img in pneumonia_train_imgs:
    origin = os.path.join('chest_xray/train/PNEUMONIA', img)
    destination = os.path.join('all_pneumonia/', img)
    shutil.copyfile(origin, destination)

# copying all the images from the original pneumonia test folder to 'all_pneumonia' folder
for img in pneumonia_test_imgs:
    origin = os.path.join('chest_xray/test/PNEUMONIA', img)
    destination = os.path.join('all_pneumonia/', img)
    shutil.copyfile(origin, destination)

# copying all the images from the original pneumonia val folder to 'all_pneumonia' folder
for img in pneumonia_val_imgs:
    origin = os.path.join('chest_xray/val/PNEUMONIA', img)
    destination = os.path.join('all_pneumonia/', img)
    shutil.copyfile(origin, destination)
```

Calculating image totals, and the ratio of 'pneumonia' to 'normal' folders.

In [12]:
```
all_normal_list = os.listdir('all_normal/')
all_pneumonia_list = os.listdir('all_pneumonia/')
print('There are ',len(all_normal_list),' total normal images')
print('There are ',len(all_pneumonia_list),' total pneumonia images')
print('There are ',len(all_normal_list) + len(all_pneumonia_list),' total images')
ratio = (round(len(all_pneumonia_list)/len(all_normal_list),2))
print('The ratio of pneumonia images to normal images is aproximately ', ratio,':1')
```

```
There are  1583  total normal images
There are  4273  total pneumonia images
There are  5856  total images
The ratio of pneumonia images to normal images is aproximately  2.7 :1
```

Calculating how many of each file will go the corresponding 'train', 'test', and 'val' folders.

In [13]:
```
print('''The train/test/val split will be approximately .8,.1,.1, with an approximate 2.7:1 ratio
between the pneumonia and normal images respectively.\n''')

print('The total number of validation images is: ',round(5856*.1))
print('The total number of normal validation images is: ',round(586/3.7))
print('The total number of pneumonia validation images is: ',round(586-158),'\n')

print('The total number of test images is: ',round(5856*.1))
print('The total number of normal test images is: ',round(586/3.7))
print('The total number of pneumonia test images is: ',round(586-158),'\n')

print('The total number of train images is: ',(5856 - (2*586)))
print('The total number of normal train images is: ',round(4684/3.7)+1)
print('The total number of pneumonia train images is: ',round(4684-1267),'\n')
```

```
The train/test/val split will be approximately .8,.1,.1, with an approximate 2.7:1 ratio
between the pneumonia and normal images respectively.

The total number of validation images is:  586
The total number of normal validation images is:  158
The total number of pneumonia validation images is:  428

The total number of test images is:  586
The total number of normal test images is:  158
The total number of pneumonia test images is:  428

The total number of train images is:  4684
The total number of normal train images is:  1267
The total number of pneumonia train images is:  3417
```

Creating lists that will be used to copythe files from the 'all_normal' and 'all_pneumonia' folders to the 'train', 'test', and'val' folders. The file names that are added to the list are chosen randomly without replacement.

In [14]:
```
random.seed(1)
val_normal_list = random.sample(all_normal_list, 158)
for img in val_normal_list:
    all_normal_list.remove(img)
print('The len of all_normal_imgs_list of removing val_normal_list images is: ', len(all_normal_list))

random.seed(2)
test_normal_list = random.sample(all_normal_list, 158)
for img in test_normal_list:
    all_normal_list.remove(img)
print('The len of all_normal_imgs_list of removing test_normal_list images is: ', len(all_normal_list))

random.seed(3)
train_normal_list = random.sample(all_normal_list, 1267)
for img in train_normal_list:
    all_normal_list.remove(img)
print('The len of all_normal_imgs_list of removing train_normal_list images is: ', len(all_normal_list))

random.seed(4)
val_pneumonia_list = random.sample(all_pneumonia_list, 428)
for img in val_pneumonia_list:
    all_pneumonia_list.remove(img)
print('The len of all_pneumonia_imgs_list after removing val_pneumonia_list images is:',len(all_pneumoni

random.seed(5)
test_pneumonia_list = random.sample(all_pneumonia_list, 428)
for img in test_pneumonia_list:
    all_pneumonia_list.remove(img)
print('The len of all_pneumonia_imgs_list after removing test_pneumonia_list images is: ', len(all_pneur

random.seed(6)
train_pneumonia_list = random.sample(all_pneumonia_list, 3417)
for img in train_pneumonia_list:
    all_pneumonia_list.remove(img)
print('The len of all_pneumonia_imgs_list after removing train_pneumonia_list images is: ', len(all_pneu
```

```
The len of all_normal_imgs_list of removing val_normal_list images is:  1425
The len of all_normal_imgs_list of removing test_normal_list images is:  1267
The len of all_normal_imgs_list of removing train_normal_list images is:  0
The len of all_pneumonia_imgs_list after removing val_pneumonia_list images is: 3845
The len of all_pneumonia_imgs_list after removing test_pneumonia_list images is:  3417
The len of all_pneumonia_imgs_list after removing train_pneumonia_list images is:  0
```

I will use the lists to copy all the files to their corresponding folder, either 'train','test', or 'val'.

In [15]:
```
# copying 'all_normal' images to normal val folder
for img in val_normal_list:
    origin = os.path.join(all_normal_dir, img)
    destination = os.path.join(val_normal_dir, img)
    shutil.copyfile(origin, destination)

# copying 'all_normal' images to normal test folder
for img in test_normal_list:
    origin = os.path.join(all_normal_dir, img)
    destination = os.path.join(test_normal_dir, img)
    shutil.copyfile(origin, destination)

# copying 'all_normal' images to normal train folder
for img in train_normal_list:
    origin = os.path.join(all_normal_dir, img)
    destination = os.path.join(train_normal_dir, img)
    shutil.copyfile(origin, destination)

# copying 'pneumonia' images to pneumonia val folder
for img in val_pneumonia_list:
    origin = os.path.join(all_pneumonia_dir, img)
    destination = os.path.join(val_pneumonia_dir, img)
    shutil.copyfile(origin, destination)

# copying 'normal' images to pneumonia test folder
for img in test_pneumonia_list:
    origin = os.path.join(all_pneumonia_dir, img)
    destination = os.path.join(test_pneumonia_dir, img)
    shutil.copyfile(origin, destination)

# copying 'normal' images to pneumonia train folder
for img in train_pneumonia_list:
    origin = os.path.join(all_pneumonia_dir, img)
    destination = os.path.join(train_pneumonia_dir, img)
    shutil.copyfile(origin, destination)
```

Confirming all the folders have the correct number of files.

In [16]:
```
# final check to make sure all the folders have the correct number of files:

print('The val normal folder has ',len(os.listdir(val_normal_dir)))
print('The val pneumonia folder has ',len(os.listdir(val_pneumonia_dir)))
print('The test normal folder has ',len(os.listdir(test_normal_dir)))
print('The test pneumonia folder has ',len(os.listdir(test_pneumonia_dir)))
print('The train normal folder has ',len(os.listdir(train_normal_dir)))
print('The train pneumonia folder has ',len(os.listdir(train_pneumonia_dir)))
```

```
The val normal folder has  158
The val pneumonia folder has  428
The test normal folder has  158
The test pneumonia folder has  428
The train normal folder has  1267
The train pneumonia folder has  3417
```

Creating instances of generators that will serve two purposes: 1. Scale all the image values to a value between 0 and 1. 2. Transfer the images from the 'train', 'test', and 'val' folders to their corresponding numpy arrays.

In [17]:
```
# get all the data in the directory train_test_val/val (586 images), and reshape them
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        val_dir,
        target_size=(64, 64), batch_size = 586)

# get all the data in the directory train_test_val/test (586 images), and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        test_dir,
        target_size=(64, 64), batch_size = 586)

# get all the data in the directory train_test_val/train (4684 images), and reshape them
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        train_dir,
        target_size=(64, 64),batch_size = 4684)
```

```
Found 586 images belonging to 2 classes.
Found 586 images belonging to 2 classes.
Found 4684 images belonging to 2 classes.
```

Using the generators to copy the images to their corresponding data sets. For each group 'train', 'test'and 'val', there is an array of images, and an array of corresponding image labels.

```
In [18]: # create the data sets
train_images, train_labels = next(train_generator)
test_images, test_labels = next(test_generator)
val_images, val_labels = next(val_generator)
```

Reviewing the shapes of the datasets that were created.

```
In [19]: # Explore your dataset again
train_img_count = train_images.shape[0]
num_px = train_images.shape[1]
test_img_count = test_images.shape[0]
val_img_count = val_images.shape[0]

print ("Number of training samples: " + str(train_img_count))
print ("Number of testing samples: " + str(test_img_count))
print ("Number of validation samples: " + str(val_img_count))
print ("train_images shape: " + str(train_images.shape))
print ("train_labels shape: " + str(train_labels.shape))
print ("test_images shape: " + str(test_images.shape))
print ("test_labels shape: " + str(test_labels.shape))
print ("val_images shape: " + str(val_images.shape))
print ("val_labels shape: " + str(val_labels.shape))
```

```
Number of training samples: 4684
Number of testing samples: 586
Number of validation samples: 586
train_images shape: (4684, 64, 64, 3)
train_labels shape: (4684, 2)
test_images shape: (586, 64, 64, 3)
test_labels shape: (586, 2)
val_images shape: (586, 64, 64, 3)
val_labels shape: (586, 2)
```

Creating three new datasets, one each for 'train', 'test', and 'val' images, where the arrays are unrowed. This is required for use with the multi-layer perceptron.

In [20]:
```
#train_img = train_images.reshape(train_images.shape[0], -1)
train_img_unrow_dataset = train_images.reshape(train_images.shape[0], -1)
test_img_unrow_dataset = test_images.reshape(test_images.shape[0], -1)
val_img_unrow_dataset = val_images.reshape(val_images.shape[0], -1)

print(train_img_unrow_dataset.shape)
print(test_img_unrow_dataset.shape)
print(val_img_unrow_dataset.shape)
```

```
(4684, 12288)
(586, 12288)
(586, 12288)
```

Repeating the process for all the labels.

In [21]:
```
train_unrow_img_labels = np.reshape(train_labels[:,0], (4684,1))
test_unrow_img_labels = np.reshape(test_labels[:,0], (586,1))
val_unrow_img_labels = np.reshape(val_labels[:,0], (586,1))

print(train_unrow_img_labels.shape)
print(test_unrow_img_labels.shape)
print(val_unrow_img_labels.shape)
```

```
(4684, 1)
(586, 1)
(586, 1)
```

5 Reveiwing the Data Before Creating Models

5.1 Checking If Data Needs to be Normalized

In [22]:
```
# summarize pixel values
print('Train images min value:', train_images.min(),'Train images max value:', train_images.max())
print('Validation images min value:', val_images.min(),'Validation images max value:', val_images.max())
print('Test images min value:', test_images.min(),'Test images max value:', test_images.max())
```

```
Train images min value: 0.0 Train images max value: 1.0
Validation images min value: 0.0 Validation images max value: 1.0
Test images min value: 0.0 Test images max value: 1.0
```

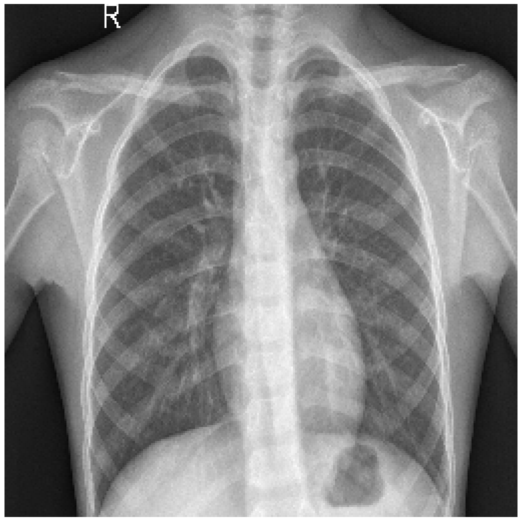
The images are already normalized.

5.2 Viewing an image

In [23]:
```
# view an image of 'normal' xray
normal_img = load_img('all_normal/IM-0001-0001.jpeg', target_size=(256, 256))
print('normal image')
normal_img
```

```
normal image
```

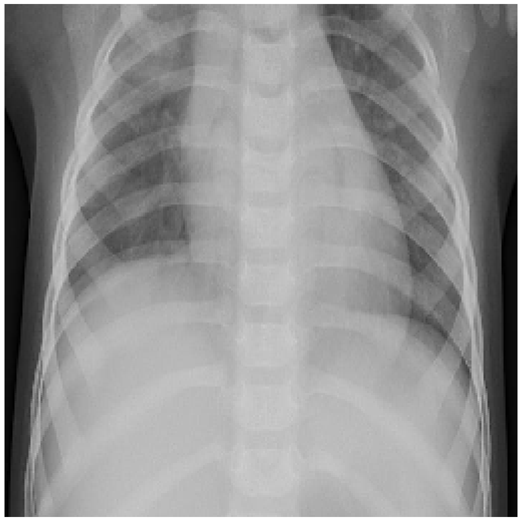
Out[23]:



In [24]:
```
# view an image of 'normal' xray
pneumonia_img = load_img('all_pneumonia/person1_bacteria_1.jpeg', target_size=(256, 256))
print('pneumonia image')
pneumonia_img
```

```
pneumonia image
```

Out[24]:



6 Neural Network models:

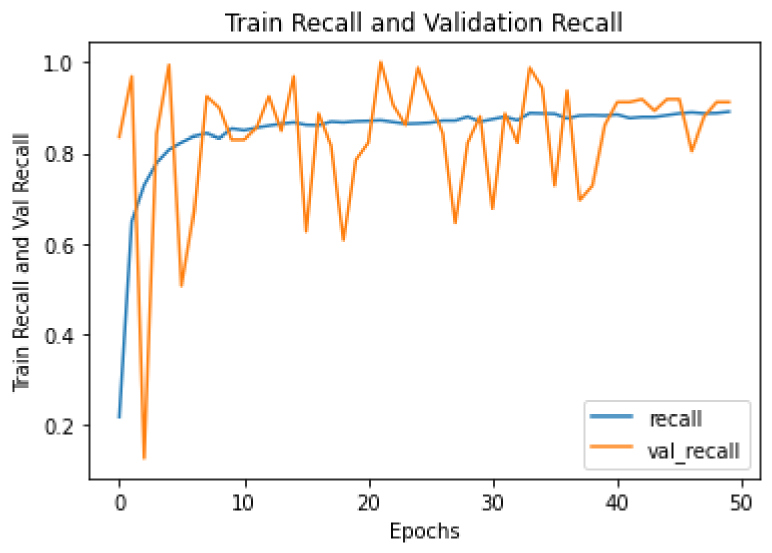
6.1 Building a multi-layer perceptron as a baseline model:

In [25]:
```
tf.random.set_seed(seed)
# Build a baseline fully connected model
from keras import models
from keras import layers

baseline_model = models.Sequential()
baseline_model.add(layers.Dense(20, activation='relu', input_shape=(12288,)))
baseline_model.add(layers.Dense(7, activation='relu'))
baseline_model.add(layers.Dense(5, activation='relu'))
baseline_model.add(layers.Dense(1, activation='sigmoid'))
baseline_model.compile(optimizer='sgd',
              loss= 'binary_crossentropy',
              metrics=['Recall'])
```

In [26]:
```
results_baseline =  baseline_model.fit(train_img_unrow_dataset,
                    train_unrow_img_labels,
                    epochs=50,
                    batch_size=32,
                    verbose=2,
                    validation_data=(val_img_unrow_dataset,val_unrow_img_labels))
```

```
Epoch 1/50
147/147 - 0s - loss: 0.5118 - recall: 0.2178 - val_loss: 0.3704 - val_recall: 0.8354
Epoch 2/50
147/147 - 0s - loss: 0.3584 - recall: 0.6488 - val_loss: 0.4929 - val_recall: 0.9684
Epoch 3/50
147/147 - 0s - loss: 0.2986 - recall: 0.7293 - val_loss: 0.6497 - val_recall: 0.1266
Epoch 4/50
147/147 - 0s - loss: 0.2630 - recall: 0.7782 - val_loss: 0.2036 - val_recall: 0.8418
Epoch 5/50
147/147 - 0s - loss: 0.2357 - recall: 0.8074 - val_loss: 0.4697 - val_recall: 0.9937
Epoch 6/50
147/147 - 0s - loss: 0.2305 - recall: 0.8232 - val_loss: 0.3269 - val_recall: 0.5063
Epoch 7/50
147/147 - 0s - loss: 0.2174 - recall: 0.8366 - val_loss: 0.2547 - val_recall: 0.6709
Epoch 8/50
147/147 - 0s - loss: 0.2069 - recall: 0.8429 - val_loss: 0.2003 - val_recall: 0.9241
Epoch 9/50
147/147 - 0s - loss: 0.2188 - recall: 0.8319 - val_loss: 0.1837 - val_recall: 0.8987
Epoch 10/50
```

```
In [27]: visualize_training_results(results_baseline)
```



```
In [28]: baseline_train_eval_dict = baseline_model.evaluate(train_img_unrow_dataset, train_unrow_img_labels, retu
         recall_dict(baseline_train_eval_dict)

Out[28]: {'recall': 0.9313}

In [29]: baseline_val_eval_dict = baseline_model.evaluate(val_img_unrow_dataset, val_unrow_img_labels,return_dict
         recall_dict(baseline_val_eval_dict)

Out[29]: {'recall': 0.9114}

In [30]: baseline_test_eval_dict = baseline_model.evaluate(test_img_unrow_dataset, test_unrow_img_labels,return_d
         recall_dict(baseline_test_eval_dict)

Out[30]: {'recall': 0.9051}
```

The recall scores are actually very good, with a test score of .9051 and a difference between the train and test scores of about only 2.5. I will try to improve this using a cnn.

6.2 Build a CNN

6.2.1 CNN-1

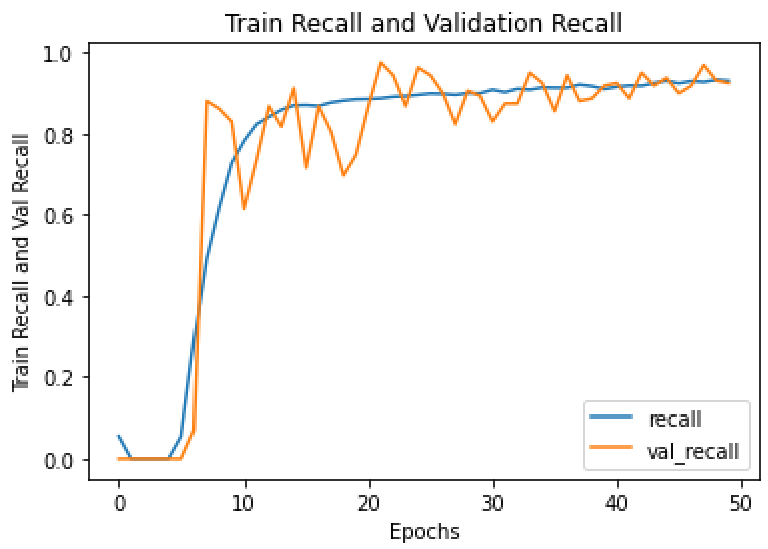
The goal here is to overfit; try to beat the ann train score of 0.9313, and try to reduce variance from there.

In [31]:
```
tf.random.set_seed(seed)

cnn_model_1 = models.Sequential()
cnn_model_1.add(layers.Conv2D(16, (2, 2), activation='relu',
                        input_shape=(64 ,64,  3)))
cnn_model_1.add(layers.MaxPooling2D((2, 2)))

cnn_model_1.add(layers.Conv2D(32, (2, 2), activation='relu'))
cnn_model_1.add(layers.MaxPooling2D((2, 2)))

cnn_model_1.add(layers.Conv2D(64, (2, 2), activation='relu'))
cnn_model_1.add(layers.MaxPooling2D((2, 2)))

cnn_model_1.add(layers.Conv2D(128, (2, 2), activation='relu'))
cnn_model_1.add(layers.MaxPooling2D((2, 2)))

cnn_model_1.add(layers.Flatten())
cnn_model_1.add(layers.Dense(64, activation='relu'))
cnn_model_1.add(layers.Dense(1, activation='sigmoid'))

cnn_model_1.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['Recall'])
```

```
In [32]: results_cnn1 = cnn_model_1.fit(train_images,
                                        train_unrow_img_labels,
                                        epochs=50,
                                        batch_size=32,
                                        verbose=2,
                                        validation_data=(val_images, val_unrow_img_labels))
```

```
Epoch 1/50
147/147 - 4s - loss: 0.6094 - recall: 0.0552 - val_loss: 0.5845 - val_recall: 0.0000e+00
Epoch 2/50
147/147 - 4s - loss: 0.5843 - recall: 0.0000e+00 - val_loss: 0.5790 - val_recall: 0.0000e+00
Epoch 3/50
147/147 - 4s - loss: 0.5771 - recall: 0.0000e+00 - val_loss: 0.5711 - val_recall: 0.0000e+00
Epoch 4/50
147/147 - 4s - loss: 0.5671 - recall: 0.0000e+00 - val_loss: 0.5568 - val_recall: 0.0000e+00
Epoch 5/50
147/147 - 4s - loss: 0.5456 - recall: 0.0000e+00 - val_loss: 0.5284 - val_recall: 0.0000e+00
Epoch 6/50
147/147 - 4s - loss: 0.5111 - recall: 0.0545 - val_loss: 0.5177 - val_recall: 0.0000e+00
Epoch 7/50
147/147 - 4s - loss: 0.4751 - recall: 0.2889 - val_loss: 0.4406 - val_recall: 0.0696
Epoch 8/50
147/147 - 4s - loss: 0.4114 - recall: 0.4909 - val_loss: 0.4079 - val_recall: 0.8797
Epoch 9/50
147/147 - 4s - loss: 0.3513 - recall: 0.6164 - val_loss: 0.3473 - val_recall: 0.8608
Epoch 10/50
```

In [33]:
```
visualize_training_results(results_cnn1)
```



In [34]:
```
cnn_1_train_eval_dict = cnn_model_1.evaluate(train_images, train_unrow_img_labels,return_dict=1, verbose
recall_dict(cnn_1_train_eval_dict)
```

Out[34]:
```
{'recall': 0.9376}
```

In [35]:
```
cnn_1_val_eval_dict = cnn_model_1.evaluate(val_images, val_unrow_img_labels, return_dict=1, verbose=0)
recall_dict(cnn_1_val_eval_dict)
```

Out[35]:
```
{'recall': 0.9241}
```

In [36]:
```
cnn_1_test_eval_dict = cnn_model_1.evaluate(test_images, test_unrow_img_labels, return_dict=1, verbose=(
recall_dict(cnn_1_test_eval_dict)
```

Out[36]:
```
{'recall': 0.8797}
```

The train score is about the same as that of the baseline. My main concern right now is trying to reduce potetntial bias so I will add more nodes to the next model and see what happens.

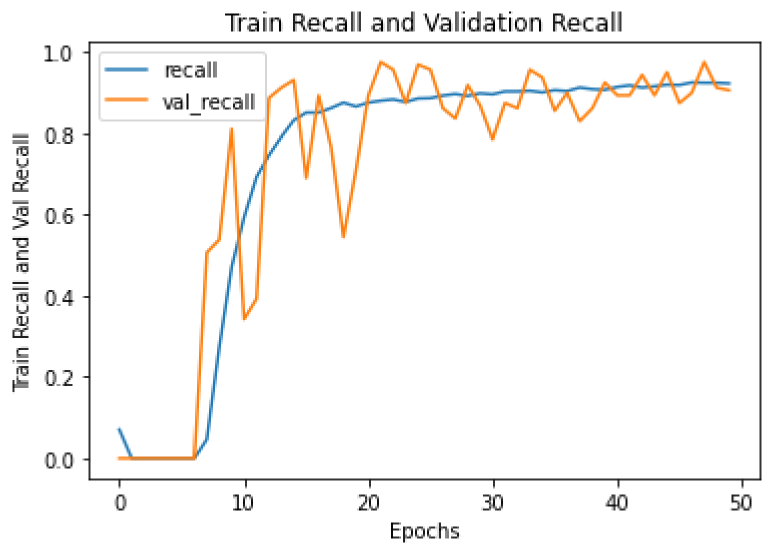
6.2.2 CNN-2

In [37]:
```
tf.random.set_seed(seed)

cnn_model_2 = models.Sequential()
cnn_model_2.add(layers.Conv2D(32, (2, 2), activation='relu',
                        input_shape=(64 ,64,  3)))
cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Conv2D(32, (2, 2), activation='relu'))
cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Conv2D(64, (2, 2), activation='relu'))
cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Conv2D(128, (2, 2), activation='relu'))
cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Flatten())
cnn_model_2.add(layers.Dense(64, activation='relu'))
cnn_model_2.add(layers.Dense(1, activation='sigmoid'))

cnn_model_2.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['Recall'])
```

In [38]:
```
results_cnn2 = cnn_model_2.fit(train_images,
                        train_unrow_img_labels,
                        epochs=50,
                        batch_size=32,
                        verbose=2,
                        validation_data=(val_images, val_unrow_img_labels))
```

```
Epoch 1/50
147/147 - 6s - loss: 0.6346 - recall: 0.0710 - val_loss: 0.5917 - val_recall: 0.0000e+00
Epoch 2/50
147/147 - 6s - loss: 0.5865 - recall: 0.0000e+00 - val_loss: 0.5820 - val_recall: 0.0000e+00
Epoch 3/50
147/147 - 6s - loss: 0.5816 - recall: 0.0000e+00 - val_loss: 0.5781 - val_recall: 0.0000e+00
Epoch 4/50
147/147 - 6s - loss: 0.5773 - recall: 0.0000e+00 - val_loss: 0.5731 - val_recall: 0.0000e+00
Epoch 5/50
147/147 - 6s - loss: 0.5702 - recall: 0.0000e+00 - val_loss: 0.5648 - val_recall: 0.0000e+00
Epoch 6/50
147/147 - 6s - loss: 0.5606 - recall: 0.0000e+00 - val_loss: 0.5531 - val_recall: 0.0000e+00
Epoch 7/50
147/147 - 6s - loss: 0.5414 - recall: 0.0000e+00 - val_loss: 0.5383 - val_recall: 0.0000e+00
Epoch 8/50
147/147 - 6s - loss: 0.5045 - recall: 0.0466 - val_loss: 0.4959 - val_recall: 0.5063
Epoch 9/50
147/147 - 6s - loss: 0.4768 - recall: 0.2723 - val_loss: 0.4255 - val_recall: 0.5380
Epoch 10/50
147/147 - 5s - loss: 0.4186 - recall: 0.4704 - val_loss: 0.3909 - val_recall: 0.8101
```

In [39]:
```
visualize_training_results(results_cnn2)
```



In [40]:
```
cnn_2_train_eval_dict = cnn_model_2.evaluate(train_images, train_unrow_img_labels,return_dict=1, verbose
recall_dict(cnn_2_train_eval_dict)
```

Out[40]:
```
{'recall': 0.9376}
```

```
In [41]: cnn_2_val_eval_dict = cnn_model_2.evaluate(val_images, val_unrow_img_labels, return_dict=1, verbose=0)
         recall_dict(cnn_2_val_eval_dict)

Out[41]: {'recall': 0.9051}
```

```
In [42]: cnn_2_test_eval_dict = cnn_model_2.evaluate(test_images, test_unrow_img_labels, return_dict=1, verbose=(
         recall_dict(cnn_2_test_eval_dict)

Out[42]: {'recall': 0.8987}
```

After adding 16 nodes to the second activation layer, the train score is virtually the same, with the recall score actually going back up relative to 'cnn1'. 93% is a pretty high score, and I am not certain I can improve on that, so I will try to reduce variance with the next model.

6.2.3 CNN-3

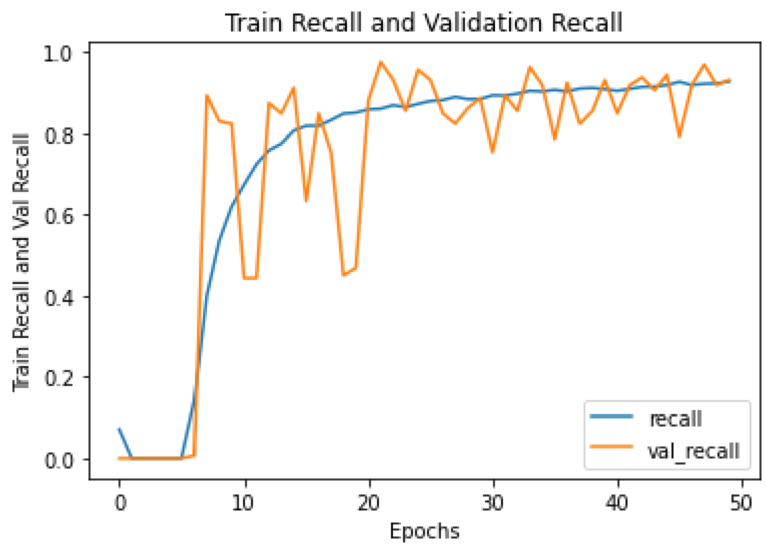
I increased the filter size in the fourth activation layer, in an attempt to increase generalizability, and therefor reduce variance.

In [43]:
```
tf.random.set_seed(seed)

cnn_model_3 = models.Sequential()
cnn_model_3.add(layers.Conv2D(32, (2, 2), activation='relu',
                        input_shape=(64 ,64,  3)))
cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Conv2D(32, (2, 2), activation='relu'))
cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Conv2D(64, (2, 2), activation='relu'))
cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Flatten())
cnn_model_3.add(layers.Dense(64, activation='relu'))
cnn_model_3.add(layers.Dense(1, activation='sigmoid'))

cnn_model_3.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['Recall'])
```

In [44]:
```
results_cnn3 = cnn_model_3.fit(train_images,
                    train_unrow_img_labels,
                    epochs=50,
                    batch_size=32,
                    verbose=2,
                    validation_data=(val_images, val_unrow_img_labels))
```

```
Epoch 1/50
147/147 - 6s - loss: 0.6321 - recall: 0.0710 - val_loss: 0.5879 - val_recall: 0.0000e+00
Epoch 2/50
147/147 - 7s - loss: 0.5841 - recall: 0.0000e+00 - val_loss: 0.5797 - val_recall: 0.0000e+00
Epoch 3/50
147/147 - 7s - loss: 0.5785 - recall: 0.0000e+00 - val_loss: 0.5743 - val_recall: 0.0000e+00
Epoch 4/50
147/147 - 6s - loss: 0.5717 - recall: 0.0000e+00 - val_loss: 0.5651 - val_recall: 0.0000e+00
Epoch 5/50
147/147 - 6s - loss: 0.5586 - recall: 0.0000e+00 - val_loss: 0.5482 - val_recall: 0.0000e+00
Epoch 6/50
147/147 - 6s - loss: 0.5353 - recall: 0.0000e+00 - val_loss: 0.5318 - val_recall: 0.0000e+00
Epoch 7/50
147/147 - 6s - loss: 0.4908 - recall: 0.1389 - val_loss: 0.4810 - val_recall: 0.0063
Epoch 8/50
147/147 - 6s - loss: 0.4422 - recall: 0.3994 - val_loss: 0.4874 - val_recall: 0.8924
Epoch 9/50
147/147 - 6s - loss: 0.3977 - recall: 0.5351 - val_loss: 0.4086 - val_recall: 0.8291
Epoch 10/50
```

In [45]:
```
visualize_training_results(results_cnn3)
```



In [46]:
```
cnn_3_train_eval_dict = cnn_model_3.evaluate(train_images, train_unrow_img_labels,return_dict=1, verbose
recall_dict(cnn_3_train_eval_dict)
```

Out[46]:
```
{'recall': 0.9463}
```

In [47]:
```
cnn_3_val_eval_dict = cnn_model_3.evaluate(val_images, val_unrow_img_labels, return_dict=1, verbose=0)
recall_dict(cnn_3_val_eval_dict)
```

Out[47]:
```
{'recall': 0.9304}
```

In [48]:
```
cnn_3_test_eval_dict = cnn_model_3.evaluate(test_images, test_unrow_img_labels, return_dict=1, verbose=(
recall_dict(cnn_3_test_eval_dict)
```

Out[48]:
```
{'recall': 0.9177}
```

Interestingly enough, this not only gave me the highest train score so far, but the highest test score as well. This leads me to belive we can still work on reducing the bias. I will keep the 3x3 filter size for the fourth activation layer on the next model, but I will increase the number of nodes on the third activation layer from 64 to 96.

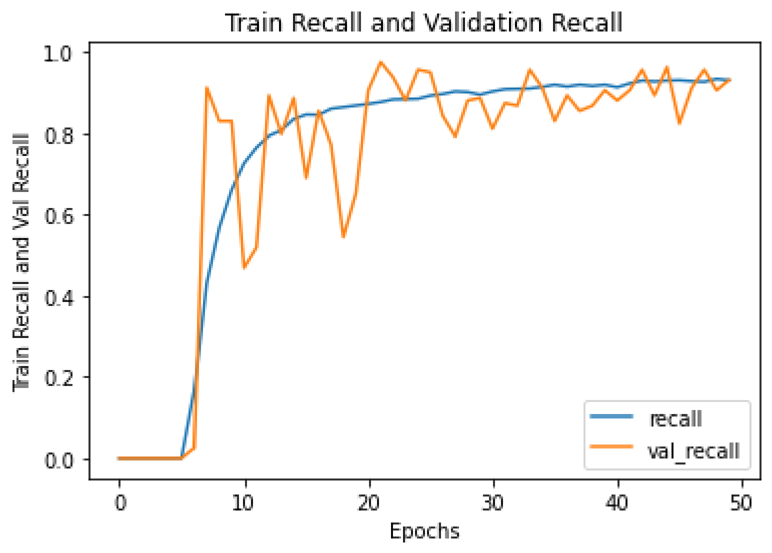
6.2.4 CNN-4

In [49]:
```
tf.random.set_seed(seed)

cnn_model_4 = models.Sequential()
cnn_model_4.add(layers.Conv2D(32, (2, 2), activation='relu',
                        input_shape=(64 ,64,  3)))
cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Conv2D(32, (2, 2), activation='relu'))
cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Conv2D(96, (2, 2), activation='relu'))
cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Flatten())
cnn_model_4.add(layers.Dense(32, activation='relu'))
cnn_model_4.add(layers.Dense(1, activation='sigmoid'))

cnn_model_4.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['Recall'])
```

In [50]:
```python
results_cnn4 = cnn_model_4.fit(train_images,
                               train_unrow_img_labels,
                               epochs=50,
                               batch_size=32,
                               verbose=2,
                               validation_data=(val_images, val_unrow_img_labels))
```

```
147/147 - 6s - loss: 0.2018 - recall: 0.8455 - val_loss: 0.2596 - val_recall: 0.6899
Epoch 17/50
147/147 - 6s - loss: 0.1930 - recall: 0.8445 - val_loss: 0.2185 - val_recall: 0.8544
Epoch 18/50
147/147 - 6s - loss: 0.1835 - recall: 0.8595 - val_loss: 0.2150 - val_recall: 0.7722
Epoch 19/50
147/147 - 6s - loss: 0.1758 - recall: 0.8635 - val_loss: 0.2975 - val_recall: 0.5443
Epoch 20/50
147/147 - 6s - loss: 0.1702 - recall: 0.8674 - val_loss: 0.2671 - val_recall: 0.6519
Epoch 21/50
147/147 - 6s - loss: 0.1662 - recall: 0.8713 - val_loss: 0.2027 - val_recall: 0.9051
Epoch 22/50
147/147 - 6s - loss: 0.1588 - recall: 0.8761 - val_loss: 0.3226 - val_recall: 0.9747
Epoch 23/50
147/147 - 6s - loss: 0.1585 - recall: 0.8824 - val_loss: 0.1967 - val_recall: 0.9367
Epoch 24/50
147/147 - 6s - loss: 0.1546 - recall: 0.8832 - val_loss: 0.1670 - val_recall: 0.8797
Epoch 25/50
147/147 - 6s - loss: 0.1477 - recall: 0.8840 - val_loss: 0.2178 - val_recall: 0.9557
Epoch 26/50
```

In [51]:
```
visualize_training_results(results_cnn4)
```



In [52]:
```
cnn_4_train_eval_dict = cnn_model_4.evaluate(train_images, train_unrow_img_labels,return_dict=1, verbose
recall_dict(cnn_4_train_eval_dict)
```

Out[52]:
```
{'recall': 0.9605}
```

```
In [53]: round(cnn_4_train_eval_dict['recall'],4)

Out[53]: 0.9605

In [54]: cnn_4_val_eval_dict = cnn_model_4.evaluate(val_images, val_unrow_img_labels, return_dict=1, verbose=0)
         recall_dict(cnn_4_val_eval_dict)

Out[54]: {'recall': 0.9304}

In [55]: cnn_4_test_eval_dict = cnn_model_4.evaluate(test_images, test_unrow_img_labels, return_dict=1, verbose=(
         recall_dict(cnn_4_test_eval_dict)

Out[55]: {'recall': 0.9304}
```

These are the best scores so far. Both the train and test scores have risen about 1.5 points, with the train scoreat over 96%. I am satisfied with this score. I will try to reduce variance on the next model by introducing a small regularization adjustment, by applying a dropout layer of .1 to the fourth activation layer (128 nodes).

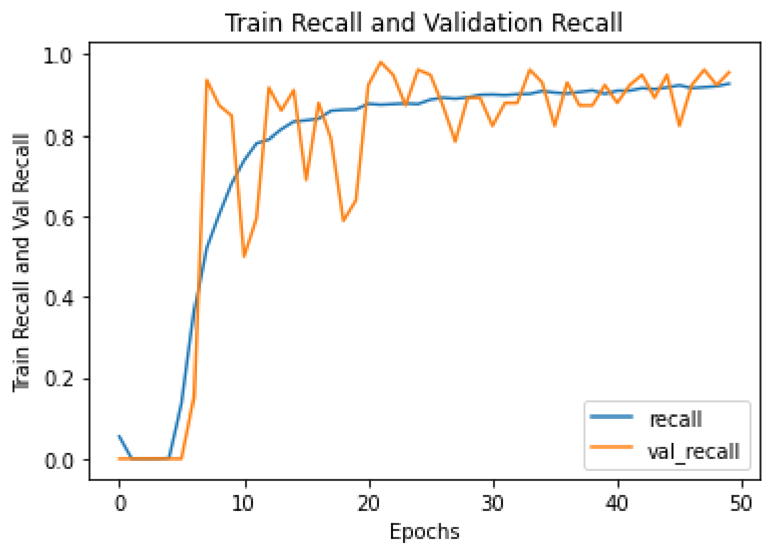
6.2.5 CNN-5

In [56]:
```
tf.random.set_seed(seed)

cnn_model_5 = models.Sequential()
cnn_model_5.add(layers.Conv2D(32, (2, 2), activation='relu',
                        input_shape=(64 ,64,  3)))
cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Conv2D(32, (2, 2), activation='relu'))
cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Conv2D(96, (2, 2), activation='relu'))
cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn_model_5.add(layers.Dropout(0.1))
cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Flatten())
cnn_model_5.add(layers.Dense(64, activation='relu'))
cnn_model_5.add(layers.Dense(1, activation='sigmoid'))

cnn_model_5.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['Recall'])
```

In [57]:
```
results_cnn5 = cnn_model_5.fit(train_images,
                    train_unrow_img_labels,
                    epochs=50,
                    batch_size=32,
                    verbose=2,
                    validation_data=(val_images, val_unrow_img_labels))
```

```
Epoch 41/50
147/147 - 7s - loss: 0.1148 - recall: 0.9100 - val_loss: 0.1333 - val_recall: 0.8797
Epoch 42/50
147/147 - 7s - loss: 0.1133 - recall: 0.9100 - val_loss: 0.1295 - val_recall: 0.9241
Epoch 43/50
147/147 - 7s - loss: 0.1083 - recall: 0.9163 - val_loss: 0.1464 - val_recall: 0.9494
Epoch 44/50
147/147 - 7s - loss: 0.1099 - recall: 0.9140 - val_loss: 0.1228 - val_recall: 0.8924
Epoch 45/50
147/147 - 7s - loss: 0.1063 - recall: 0.9179 - val_loss: 0.1537 - val_recall: 0.9494
Epoch 46/50
147/147 - 7s - loss: 0.1030 - recall: 0.9234 - val_loss: 0.1578 - val_recall: 0.8228
Epoch 47/50
147/147 - 7s - loss: 0.1051 - recall: 0.9163 - val_loss: 0.1229 - val_recall: 0.9241
Epoch 48/50
147/147 - 7s - loss: 0.1033 - recall: 0.9187 - val_loss: 0.1887 - val_recall: 0.9620
Epoch 49/50
147/147 - 7s - loss: 0.1012 - recall: 0.9211 - val_loss: 0.1230 - val_recall: 0.9241
Epoch 50/50
147/147 - 7s - loss: 0.1020 - recall: 0.9274 - val_loss: 0.1520 - val_recall: 0.9557
```

In [58]:
```
visualize_training_results(results_cnn5)
```



In [59]:
```
cnn_5_train_eval_dict = cnn_model_5.evaluate(train_images, train_unrow_img_labels,return_dict=1, verbose
recall_dict(cnn_5_train_eval_dict)
```

Out[59]:
```
{'recall': 0.9669}
```

```
In [60]: cnn_5_val_eval_dict = cnn_model_5.evaluate(val_images, val_unrow_img_labels,return_dict=1, verbose=0)
         recall_dict(cnn_5_val_eval_dict)

Out[60]: {'recall': 0.9557}
```

```
In [61]: cnn_5_test_eval_dict = cnn_model_5.evaluate(test_images, test_unrow_img_labels,return_dict=1, verbose=0)
         recall_dict(cnn_5_test_eval_dict)

Out[61]: {'recall': 0.943}
```

This model returned the best scores, making a small improvement on the train score, and increasing by aproximately 1.5 points, relative to 'CNN-4'

7 Choosing Best Model:

The best_model() function returns a dataframe sorted by auc score. The model with the highest auc score is the model with the best bias-variance balance, and therefore the best model. In this case the best model is cnn_model_5, with a test score of 0.943038, and a train-test difference of 0.023813.

```
In [62]: scores_df_sorted = best_model(6)
         scores_df_sorted

Out[62]:
```

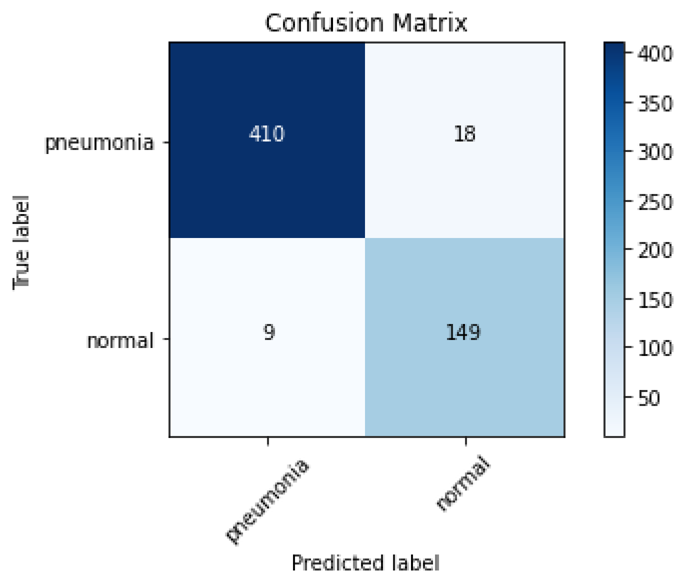
	model name	train score	test score	train-test diff	auc score
5	cnn_model_5	0.966851	0.943038	0.023813	0.959613
4	cnn_model_4	0.960537	0.930380	0.030157	0.950111
3	cnn_model_3	0.946330	0.917722	0.028608	0.944557
0	baseline_model	0.931334	0.905063	0.026271	0.939396
2	cnn_model_2	0.937648	0.898734	0.038914	0.929910
1	cnn_model_1	0.937648	0.879747	0.057901	0.910923

8 Best Model Confusion Matrix:

I created a confusion matrix plot for the presentation, as well as an easy way to calculate the accuracy score. With 559 0ut of 586 images correctly classified, this model has an accuracy score of over 95%.

In [63]:
```
tf.random.set_seed(seed)
# creating the list of predictions and rounding to 0 or 1
cnn_model_5_preds = cnn_model_5.predict(test_images)
best_model_rounded_preds  = np.round(cnn_model_5_preds)

#The following are the arguments required to create a visual plot of the confusion matrix:
# scikit-learn confusion matrix returns a numerical array with: tp, fp, tn, and fn
cm = confusion_matrix(y_true=test_unrow_img_labels, y_pred=best_model_rounded_preds)
# list with the plot labels (required as an argument)
cm_plot_labels = ['pneumonia','normal']


# plotting confusion matrix:
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
```



9 Project Conclusion: Possible Further Steps

1. Run the model on a larger dataset
2. Run the model on a balanced datset
3. Use SMOTE or an image generator to attempt to balance the training data
4. Create a custom metric that would take into account both precision and accuracy, but give priority to precision