Final Project Submission Please fill out: Student name: Dhruv Ragunathan Student pace: part time Scheduled project review date/time: 11/16/2024 Instructor name: Mark Barbour Blog post URL:

Background

Pneumonia is an infection that inflames the lungs from many potential vectors: Bacteria, Viruses, Fungi, etc. Pneumonia is a potentially fatal infection, that if not identified early on could put the patient's life at risk especially in acute-care settings.

Data sets off lungs with Pneumonia and healthy lungs were provided by Guangzhou Women and Children's Medical Center, Guangzhou.

Figure S6. Illustrative Examples of Chest X-Rays in Patients with Pneumonia, Related to Figure 6 The normal chest X-ray (left panel) depicts clear lungs without any areas of abnormal opacification in the image. Bacterial pneumonia (middle) typically exhibits a focal lobar consolidation, in this case in the right upper lobe (white arrows), whereas viral pneumonia (right) manifests with a more diffuse "interstitial" pattern in both lungs.

The link to download the dataset is here. The dataset was downloaded from this link

https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia)

Business Objectives

The goal is to create a model that accurately predicts if an X-ray of a lung has pneumonia. The model will optimize on accuracy because we can save lives by accurately predicting lungs that have pneumonia.

The model will be deployed in a hospital where time is critical. Identifying someone who has pneumonia hours before complications occur can save lives.

Data Exploration

Data contains 3 data sets, train, test, and validation. In each file, there are two folders: Pneumonia and normal. Images in the Pneumonia file are named depending on whether the source off the infection is bacterial or viral.

First separate them out based on whether they are normal or have pneumonia.

The images that contain pneumonia are divided based on whether the cause off the ailment was bacterial or viral. Since the purpose of the model is too determine whether Pneumonia is present, we will not distinguish or aim to predict the source off the pneumonia.

Google Colab Code

This is the code ran on to import the files into google colab. It is commented out for convenience.

In [ ]:
```
"""

from google.colab import drive
drive.mount('/content/gdrive')

"""
```

```
Mounted at /content/gdrive
```

In [ ]:
```
#%cd gdrive/My Drive/Pneumonia_Image_Detection/Pneumonia_Image_Detec
```

```
/content/gdrive/My Drive/Pneumonia_Image_Detection/Pneumonia_Image_Dete
ction
```

In [ ]:
```
# Import data sets

import os, shutil
import time
import matplotlib.pyplot as plt
import pandas as pd
import scipy
import numpy as np
import seaborn as sns
from PIL import Image
from scipy import ndimage
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator, array_to_i
from keras import models
from keras import layers
from keras.preprocessing import image
from sklearn.model_selection import train_test_split
from sklearn. metrics import recall_score, confusion_matrix
np.random.seed(42)
```

In [ ]:
```
folders = {}
train = {}
test = {}
test = {}
val = {}

main_folder = "chest_xray"

train_folder = main_folder + "/train"
train_P = train_folder + "/PNEUMONIA"
train_N = train_folder + "/NORMAL"

test_folder = main_folder + "/test"
test_P = test_folder + "/PNEUMONIA"
test_N = test_folder + "/NORMAL"

val_folder = main_folder + "/val"
val_P = val_folder + "/PNEUMONIA"
val_N = val_folder + "/NORMAL"


pneu_folds = [train_P,test_P,val_P] #put all pneumonia file paths in

normal_folds = [train_N,test_N,val_N] #Put all normal file paths in

all_folds = [pneu_folds,normal_folds]
```

In [ ]:

```
"""

The code below looks at all the file paths. It counts the number of
the total number of images with pneumonia or that are normal.

"""



total_normal = 0
total_pneumonia = 0

total_train_P = 0
total_train_N = 0

total_test_P = 0
total_test_N = 0

total_val_P = 0
total_val_N = 0

for folders in all_folds:

    for file_path in folders:

        folder = file_path.split("/") #Split the file path based on

        folder_name = folder[2] # Tells us if the folder is for PNEU
        folder_data = folder[1] # Tells us if the folder is for the

        if folder_name == 'PNEUMONIA':

            num_imgs = len([file for file in os.listdir(file_path) i


            total_pneumonia = total_pneumonia + num_imgs

            if folder_data == 'train':

                total_train_P = num_imgs

            if folder_data == 'test':

                total_test_P = num_imgs


            if folder_data == 'val':

                total_val_P = num_imgs


        if folder_name == 'NORMAL':

            num_imgs = len([file for file in os.listdir(file_path) i

            total_normal = total_normal + num_imgs

```

```
            if folder_data == 'train':

                total_train_N = num_imgs

            if folder_data == 'test':

                total_test_N = num_imgs


            if folder_data == 'val':

                total_val_N = num_imgs


total_images = total_pneumonia + total_normal



```

In [ ]:
```
print('There are',total_images, 'images total')

print('There are',total_pneumonia, 'lungs with pneumonia')

print('There are',total_normal, 'lungs without pneumonia')
```

```
There are 5856 images total
There are 4273 lungs with pneumonia
There are 1583 lungs without pneumonia
```

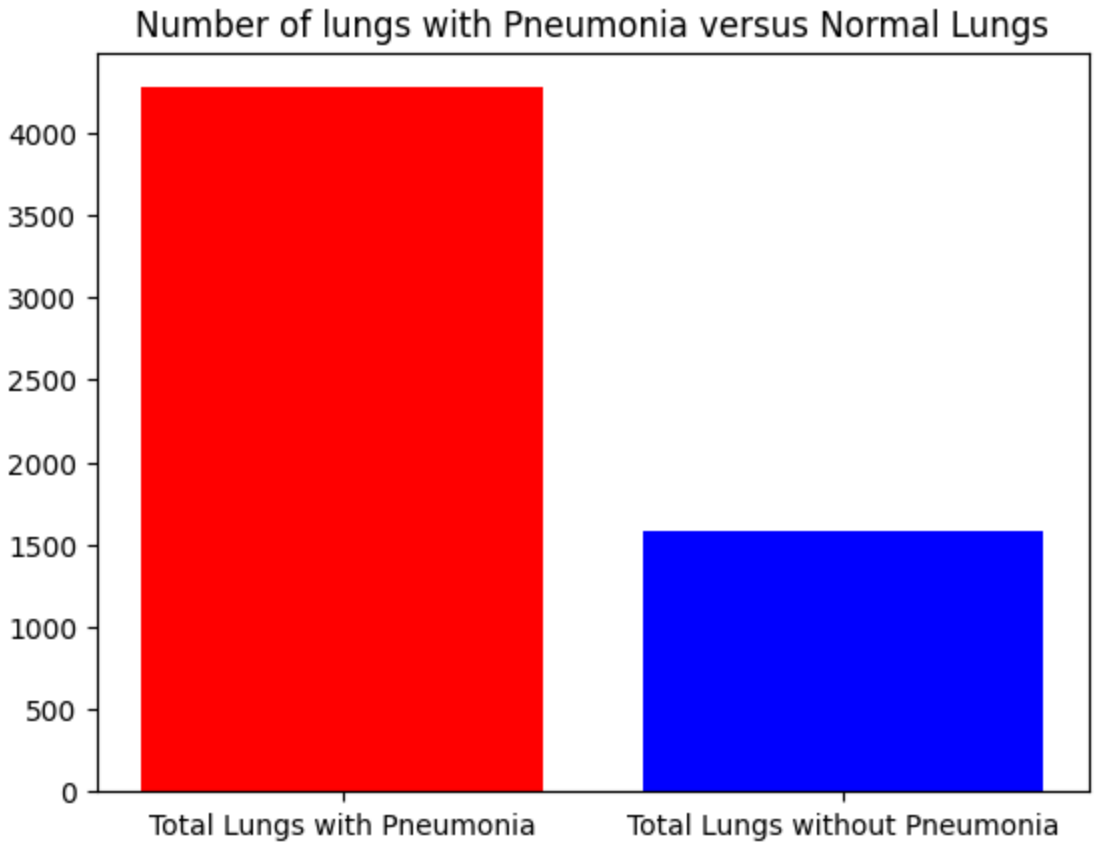
In [ ]:
```
total_train_P
total_train_N

total_test_P
total_test_N

total_val_P
total_val_N


print('In the train data set the split between pneumonia and normal
print('In the test data set the split between pneumonia and normal l
print('In the validation data set the split between pneumonia and no
```

```
In the train data set the split between pneumonia and normal lungs is
3875  and  1341  respectively.
In the test data set the split between pneumonia and normal lungs is  3
90  and  234  respectively.
In the validation data set the split between pneumonia and normal lungs
is  8  and  8  respectively.
```

The split between the training, testing, and validation data is not ideal. The data in the validation dataset is miniscule (16) compared to the training and testing set (~5200 and ~600, respectively). Later we will combine the data before doing train-test splits.

In [ ]:
```
# Visualize total number of infected lungs versus uninfected

plt.title("Number of lungs with Pneumonia versus Normal Lungs")
plt.bar(["Total Lungs with Pneumonia", "Total Lungs without Pneumoni
```

Out[8]: <BarContainer object of 2 artists>



The number of lungs with pneumonia is nearly 3 times greater than the number of lungs without. This means that if a model predicted that every image had pneumonia on this data set, it would be accurate 75% of the time. This context is important when evaluating the models in the latter part of the notebook.

Data Preparation

The Process to prepare the data was as follows:

1. First we used Image Data Generator on the images in the test, train, and val data.
2. Then we created the labels for the target variable. Whether an xray is for a patient with Pneumonia or not.
3. Due to the disparities between the given training, testing, and val set, all data was combined into one dataset.
4. Data then split into training and testing sets. Since Neural Networks are computationally expensive. Two train test splits were done too reduce the size. The final training data set contained nearly 2500 images.

In [ ]:
```
# get all the data in the directory split/test (624 images), and res
test_generator = ImageDataGenerator(rescale=1./255).flow_from_direct
        test_folder,
         batch_size = 624)

# get all the data in the directory split/validation (16 images), an
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directo
        val_folder,
        batch_size = 16)

# get all the data in the directory split/train (5216 images), and r
train_generator = ImageDataGenerator(rescale=1./255).flow_from_direc
        train_folder,
        batch_size=5216)
```

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

In [ ]:
```
# 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)
```

In [ ]:
```
# Explore your dataset again
m_train = train_images.shape[0]
num_px = train_images.shape[1]
m_test = test_images.shape[0]
m_val = val_images.shape[0]

print ("Number of training samples: " + str(m_train))
print ("Number of testing samples: " + str(m_test))
print ("Number of validation samples: " + str(m_val))
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: 5216
Number of testing samples: 624
Number of validation samples: 16
train_images shape: (5216, 256, 256, 3)
train_labels shape: (5216, 2)
test_images shape: (624, 256, 256, 3)
test_labels shape: (624, 2)
val_images shape: (16, 256, 256, 3)
val_labels shape: (16, 2)
```

In [ ]:
```
# combine three original datasets into one for data and labels
X = np.concatenate((train_images, test_images, val_images))
y_labels = np.concatenate((train_labels, test_labels, val_labels))
```

In [ ]:
```
# total dataset size
X.shape
```

Out[13]:
```
(5856, 256, 256, 3)
```

In [ ]:
```
y = np.reshape(y_labels[:,0], (5856,1))
```

In [ ]:
```
# Further Explore Dataset

m_train = train_images.shape[0]
num_px = train_images.shape[1]
m_test = test_images.shape[0]
m_val = val_images.shape[0]

print ("Number of training samples: " + str(m_train))
print ("Number of testing samples: " + str(m_test))
print ("Number of validation samples: " + str(m_val))
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: 5216
Number of testing samples: 624
Number of validation samples: 16
train_images shape: (5216, 256, 256, 3)
train_labels shape: (5216, 2)
test_images shape: (624, 256, 256, 3)
test_labels shape: (624, 2)
val_images shape: (16, 256, 256, 3)
val_labels shape: (16, 2)
```

Train - Test Data Split

Using the full training data is not ideal to iteratively determine model parameters. The complete training set takes around 30 minutes to run with straightforward parameters (i.e. one convulution layer, one pooling layer, and one dense layer).

The strategy here is too split the data set into a smaller chunk.

Convolutional Neural Network Data Preparation

Now we need to do a train test split to the data formatted for the Convolutional Neural Network.

In [ ]:
```
# First split the data into half.

X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X,y, tes
```

In [ ]:
```
# Then split again by 15%

X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_train_
```

Modeling

First Convolution Neural Network

Base model is a simple convolution neural network. Contains one convolution layer, one pooling layer, and 10 neurons in the dense layer.

This model has achieved a ~96% accuracy on the training data set and a 95% accuracy on the testing data set.

In [ ]:
```
model_cnn_1 = models.Sequential()

# add convolutional and pooling layers
model_cnn_1.add(layers.Conv2D(8, (4, 4), activation='relu',
                              input_shape=(256 ,256,  3)))
model_cnn_1.add(layers.MaxPooling2D((2, 2)))

# flatten pooled featrue maps
model_cnn_1.add(layers.Flatten())

# add dense hidden layer and output
model_cnn_1.add(layers.Dense(10, activation='relu'))
model_cnn_1.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# compile model
model_cnn_1.compile(optimizer='sgd',
                    loss='binary_crossentropy',
                    metrics=['accuracy','Recall'])
```

In [ ]:

```
# fit model to training data and validate
cnn_history_1 = model_cnn_1.fit(X_train_2,
                    y_train_2,
                    epochs=30,
                    batch_size=32,
                    validation_data=(X_test_2, y_test_2))
```

```
Epoch 1/30
78/78 [==============================] - 25s 311ms/step - loss: 0.5903
- accuracy: 0.7062 - recall: 0.1843 - val_loss: 0.4615 - val_accuracy:
0.7523 - val_recall: 0.2270
Epoch 2/30
78/78 [==============================] - 22s 283ms/step - loss: 0.4543
- accuracy: 0.7890 - recall: 0.4758 - val_loss: 0.3119 - val_accuracy:
0.8636 - val_recall: 0.5957
Epoch 3/30
78/78 [==============================] - 23s 291ms/step - loss: 0.3088
- accuracy: 0.8742 - recall: 0.7296 - val_loss: 0.2503 - val_accuracy:
0.8977 - val_recall: 0.7163
Epoch 4/30
78/78 [==============================] - 26s 329ms/step - loss: 0.2775
- accuracy: 0.8879 - recall: 0.7281 - val_loss: 0.1984 - val_accuracy:
0.9227 - val_recall: 0.8582
Epoch 5/30
78/78 [==============================] - 23s 290ms/step - loss: 0.2105
- accuracy: 0.9216 - recall: 0.8384 - val_loss: 0.2769 - val_accuracy:
0.8818 - val_recall: 0.6667
Epoch 6/30
78/78 [==============================] - 23s 297ms/step - loss: 0.1935
- accuracy: 0.9236 - recall: 0.8459 - val_loss: 0.3137 - val_accuracy:
0.8659 - val_recall: 0.9645
Epoch 7/30
78/78 [==============================] - 22s 288ms/step - loss: 0.1828
- accuracy: 0.9317 - recall: 0.8580 - val_loss: 0.1908 - val_accuracy:
0.9318 - val_recall: 0.8369
Epoch 8/30
78/78 [==============================] - 23s 298ms/step - loss: 0.1765
- accuracy: 0.9345 - recall: 0.8610 - val_loss: 0.2001 - val_accuracy:
0.9341 - val_recall: 0.8298
Epoch 9/30
78/78 [==============================] - 22s 283ms/step - loss: 0.1720
- accuracy: 0.9373 - recall: 0.8716 - val_loss: 0.2110 - val_accuracy:
0.9068 - val_recall: 0.9220
Epoch 10/30
78/78 [==============================] - 22s 286ms/step - loss: 0.1650
- accuracy: 0.9361 - recall: 0.8761 - val_loss: 0.1843 - val_accuracy:
0.9341 - val_recall: 0.8936
Epoch 11/30
78/78 [==============================] - 23s 298ms/step - loss: 0.1579
- accuracy: 0.9397 - recall: 0.8761 - val_loss: 0.1867 - val_accuracy:
0.9318 - val_recall: 0.8298
Epoch 12/30
78/78 [==============================] - 22s 287ms/step - loss: 0.1748
- accuracy: 0.9369 - recall: 0.8686 - val_loss: 0.1863 - val_accuracy:
0.9341 - val_recall: 0.8440
Epoch 13/30
78/78 [==============================] - 27s 344ms/step - loss: 0.1430
- accuracy: 0.9461 - recall: 0.8912 - val_loss: 0.1870 - val_accuracy:
0.9227 - val_recall: 0.9220
Epoch 14/30
78/78 [==============================] - 23s 289ms/step - loss: 0.1443
- accuracy: 0.9494 - recall: 0.8958 - val_loss: 0.3185 - val_accuracy:
0.8886 - val_recall: 0.6809
Epoch 15/30
```

```
78/78 [==============================] - 23s 299ms/step - loss: 0.1425
- accuracy: 0.9473 - recall: 0.8852 - val_loss: 0.1904 - val_accuracy:
0.9341 - val_recall: 0.8298
Epoch 16/30
78/78 [==============================] - 22s 287ms/step - loss: 0.1258
- accuracy: 0.9558 - recall: 0.9154 - val_loss: 0.2753 - val_accuracy:
0.8955 - val_recall: 0.9716
Epoch 17/30
78/78 [==============================] - 23s 294ms/step - loss: 0.1281
- accuracy: 0.9546 - recall: 0.9079 - val_loss: 0.1642 - val_accuracy:
0.9341 - val_recall: 0.8865
Epoch 18/30
78/78 [==============================] - 23s 294ms/step - loss: 0.1227
- accuracy: 0.9558 - recall: 0.9094 - val_loss: 0.1624 - val_accuracy:
0.9386 - val_recall: 0.8794
Epoch 19/30
78/78 [==============================] - 22s 287ms/step - loss: 0.1201
- accuracy: 0.9562 - recall: 0.9063 - val_loss: 0.2227 - val_accuracy:
0.9068 - val_recall: 0.7234
Epoch 20/30
78/78 [==============================] - 23s 297ms/step - loss: 0.1151
- accuracy: 0.9538 - recall: 0.9094 - val_loss: 0.2273 - val_accuracy:
0.9182 - val_recall: 0.7801
Epoch 21/30
78/78 [==============================] - 22s 286ms/step - loss: 0.1058
- accuracy: 0.9618 - recall: 0.9245 - val_loss: 0.1560 - val_accuracy:
0.9386 - val_recall: 0.8936
Epoch 22/30
78/78 [==============================] - 23s 295ms/step - loss: 0.1025
- accuracy: 0.9606 - recall: 0.9215 - val_loss: 0.1841 - val_accuracy:
0.9409 - val_recall: 0.8511
Epoch 23/30
78/78 [==============================] - 23s 291ms/step - loss: 0.1021
- accuracy: 0.9630 - recall: 0.9305 - val_loss: 0.1553 - val_accuracy:
0.9364 - val_recall: 0.9149
Epoch 24/30
78/78 [==============================] - 22s 288ms/step - loss: 0.0992
- accuracy: 0.9670 - recall: 0.9290 - val_loss: 0.1713 - val_accuracy:
0.9432 - val_recall: 0.8582
Epoch 25/30
78/78 [==============================] - 31s 398ms/step - loss: 0.0916
- accuracy: 0.9715 - recall: 0.9411 - val_loss: 0.1504 - val_accuracy:
0.9432 - val_recall: 0.8865
Epoch 26/30
78/78 [==============================] - 32s 410ms/step - loss: 0.0868
- accuracy: 0.9682 - recall: 0.9426 - val_loss: 0.1511 - val_accuracy:
0.9364 - val_recall: 0.9291
Epoch 27/30
78/78 [==============================] - 23s 289ms/step - loss: 0.0831
- accuracy: 0.9723 - recall: 0.9486 - val_loss: 0.1697 - val_accuracy:
0.9432 - val_recall: 0.8652
Epoch 28/30
78/78 [==============================] - 24s 304ms/step - loss: 0.0807
- accuracy: 0.9719 - recall: 0.9411 - val_loss: 0.1523 - val_accuracy:
0.9409 - val_recall: 0.9007
Epoch 29/30
78/78 [==============================] - 23s 290ms/step - loss: 0.0756
```

```
- accuracy: 0.9763 - recall: 0.9517 - val_loss: 0.1576 - val_accuracy:
0.9364 - val_recall: 0.9078
Epoch 30/30
78/78 [==============================] - 23s 295ms/step - loss: 0.0784
- accuracy: 0.9747 - recall: 0.9547 - val_loss: 0.1538 - val_accuracy:
0.9409 - val_recall: 0.8723
```

In [ ]:
```
#model_cnn_1.save('Models/model_cnn_1.keras')
```

In [ ]:
```
model_cnn_1 = models.load_model('Models/model_cnn_1.keras')
```

In [ ]:
```
conv_results_train_1 = model_cnn_1.evaluate(X_train_2, y_train_2)
```

```
78/78 [==============================] - 6s 73ms/step - loss: 0.0609 -
accuracy: 0.9847 - recall: 0.9577
```

In [ ]:
```
conv_results_test_1 = model_cnn_1.evaluate(X_test_2, y_test_2)
```

```
14/14 [==============================] - 1s 76ms/step - loss: 0.1538 -
accuracy: 0.9409 - recall: 0.8723
```

In [ ]:
```
cnn_1_preds = model_cnn_1.predict(X_test_2)


```

```
14/14 [==============================] - 1s 76ms/step
```

In [ ]:
```
conv_results_test_1
```

Out[22]:
```
[0.15380030870437622, 0.9409090876579285, 0.8723404407501221]
```

This model has achieved a ~97% accuracy on the training data set, a 94% accuracy on the testing data set, and a recall off 87% on the testing data set.

The high and consistent accuracy between the training data set and testing dataset and the high recall indicates that the model performed well.

In [ ]:

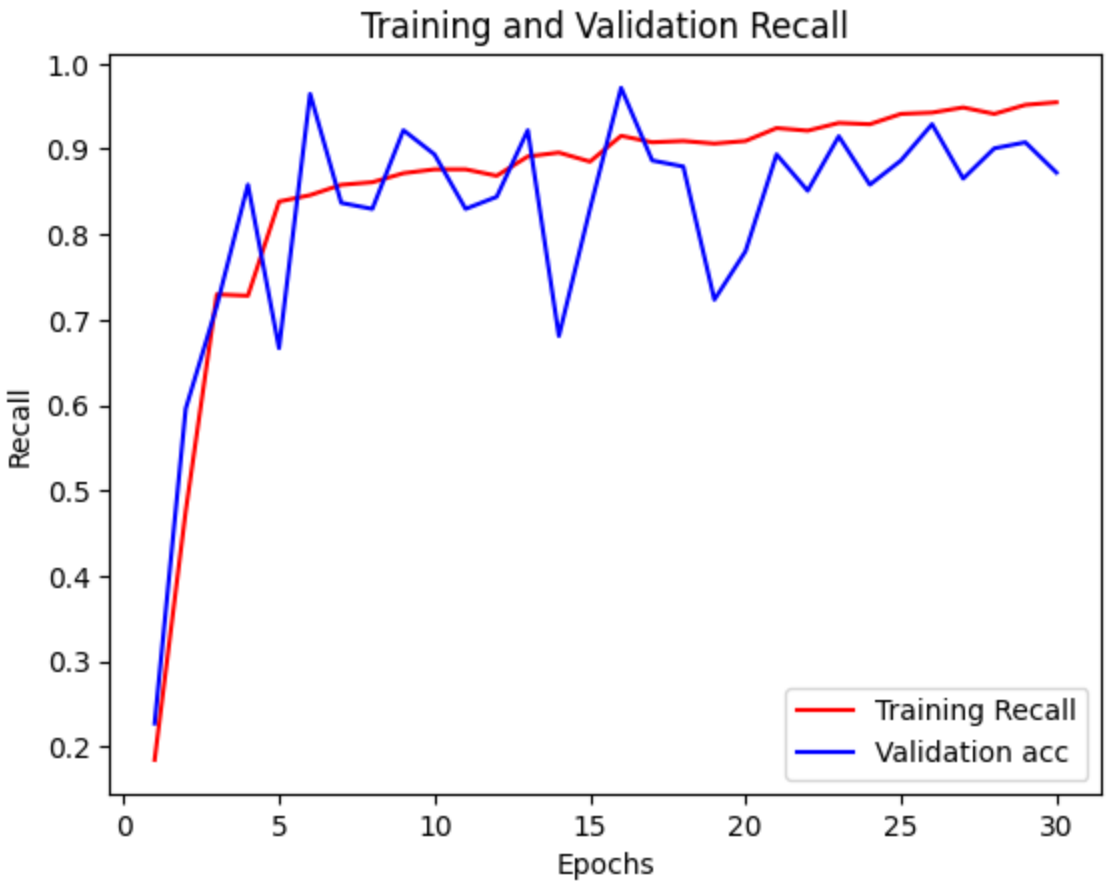
```
cnn_model_1_val_dict = cnn_history_1.history
loss_values1 = cnn_model_1_val_dict["loss"]
val_loss_values1= cnn_model_1_val_dict["val_loss"]

epochs = range(1,len(loss_values1) + 1)
plt.plot(epochs, loss_values1, "g", label="Training loss")
plt.plot(epochs, val_loss_values1, "blue", label="Validation loss")

plt.title("Training and Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-fdb6b0548b6b> in <cell line: 1>()
----> 1 cnn_model_1_val_dict = cnn_history_1.history
      2 loss_values1 = cnn_model_1_val_dict["loss"]
      3 val_loss_values1= cnn_model_1_val_dict["val_loss"]
      4
      5 epochs = range(1,len(loss_values1) + 1)

NameError: name 'cnn_history_1' is not defined
```

In [ ]:

```
acc_values1 = cnn_model_1_val_dict["accuracy"]
val_acc_values1 = cnn_model_1_val_dict["val_accuracy"]

plt.plot(epochs, acc_values1, "r", label="Training acc")
plt.plot(epochs, val_acc_values1, "blue", label="Validation acc")

plt.title("Training and Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

In [ ]:
```
Recall_values1 = cnn_model_1_val_dict["recall"]
val_Recall_values1 = cnn_model_1_val_dict["val_recall"]

plt.plot(epochs, Recall_values1, "r", label="Training Recall")
plt.plot(epochs, val_Recall_values1, "blue", label="Validation acc")

plt.title("Training and Validation Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.legend()
plt.show()
```



Second Colvolutional Neural Networks

- Add two more Convolutional Layers
- Increase Dense Layer to 30 neurons

Accuracy of training and testing data is 96%.

In [ ]:
```
model_cnn_2 = models.Sequential()

# add 3 convolutional and pooling layers
model_cnn_2.add(layers.Conv2D(8, (4, 4), activation='relu',
                        input_shape=(256 ,256,  3)))
model_cnn_2.add(layers.MaxPooling2D((2, 2)))

# Second Convolutional Layer

model_cnn_2.add(layers.Conv2D(8, (4, 4), activation='relu',
                        input_shape=(256 ,256,  3)))
model_cnn_2.add(layers.MaxPooling2D((2, 2)))

#Third Convolutional Layer

model_cnn_2.add(layers.Conv2D(8, (4, 4), activation='relu',
                        input_shape=(256 ,256,  3)))
model_cnn_2.add(layers.MaxPooling2D((2, 2)))

# flatten pooled featrue maps
model_cnn_2.add(layers.Flatten())

# add dense hidden layer and output
model_cnn_2.add(layers.Dense(30, activation='relu'))
model_cnn_2.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# compile model
model_cnn_2.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy','Recall'])
```

In [ ]:

```
# fit model to training data and validate
cnn_history_2 = model_cnn_2.fit(X_train_2,
                    y_train_2,
                    epochs=30,
                    batch_size=32,
                    validation_data=(X_test_2, y_test_2))
```

```
Epoch 1/30
78/78 [==============================] - 35s 444ms/step - loss: 0.5688
- accuracy: 0.7335 - recall: 0.0740 - val_loss: 0.5363 - val_accuracy:
0.6795 - val_recall: 0.0000e+00
Epoch 2/30
78/78 [==============================] - 34s 438ms/step - loss: 0.4726
- accuracy: 0.7777 - recall: 0.3459 - val_loss: 0.3529 - val_accuracy:
0.8523 - val_recall: 0.5674
Epoch 3/30
78/78 [==============================] - 34s 440ms/step - loss: 0.3478
- accuracy: 0.8485 - recall: 0.6390 - val_loss: 0.2499 - val_accuracy:
0.9000 - val_recall: 0.8085
Epoch 4/30
78/78 [==============================] - 34s 440ms/step - loss: 0.2893
- accuracy: 0.8883 - recall: 0.7523 - val_loss: 0.2869 - val_accuracy:
0.8818 - val_recall: 0.9433
Epoch 5/30
78/78 [==============================] - 34s 439ms/step - loss: 0.2265
- accuracy: 0.9096 - recall: 0.8036 - val_loss: 0.2054 - val_accuracy:
0.9182 - val_recall: 0.8511
Epoch 6/30
78/78 [==============================] - 34s 441ms/step - loss: 0.2251
- accuracy: 0.9136 - recall: 0.8172 - val_loss: 0.2764 - val_accuracy:
0.9068 - val_recall: 0.9504
Epoch 7/30
78/78 [==============================] - 34s 438ms/step - loss: 0.2008
- accuracy: 0.9216 - recall: 0.8414 - val_loss: 0.2103 - val_accuracy:
0.9318 - val_recall: 0.8369
Epoch 8/30
78/78 [==============================] - 34s 442ms/step - loss: 0.1901
- accuracy: 0.9277 - recall: 0.8474 - val_loss: 0.1972 - val_accuracy:
0.9318 - val_recall: 0.8440
Epoch 9/30
78/78 [==============================] - 36s 464ms/step - loss: 0.1757
- accuracy: 0.9341 - recall: 0.8595 - val_loss: 0.1935 - val_accuracy:
0.9318 - val_recall: 0.9078
Epoch 10/30
78/78 [==============================] - 34s 440ms/step - loss: 0.1834
- accuracy: 0.9329 - recall: 0.8640 - val_loss: 0.1762 - val_accuracy:
0.9364 - val_recall: 0.8936
Epoch 11/30
78/78 [==============================] - 34s 441ms/step - loss: 0.1743
- accuracy: 0.9341 - recall: 0.8565 - val_loss: 0.2441 - val_accuracy:
0.9000 - val_recall: 0.9433
Epoch 12/30
78/78 [==============================] - 34s 438ms/step - loss: 0.1629
- accuracy: 0.9401 - recall: 0.8701 - val_loss: 0.1807 - val_accuracy:
0.9364 - val_recall: 0.9362
Epoch 13/30
78/78 [==============================] - 34s 440ms/step - loss: 0.1695
- accuracy: 0.9389 - recall: 0.8822 - val_loss: 0.1846 - val_accuracy:
0.9341 - val_recall: 0.8440
Epoch 14/30
78/78 [==============================] - 34s 437ms/step - loss: 0.1578
- accuracy: 0.9385 - recall: 0.8761 - val_loss: 0.2137 - val_accuracy:
0.9364 - val_recall: 0.8511
Epoch 15/30
```

```
78/78 [==============================] - 34s 441ms/step - loss: 0.1468
- accuracy: 0.9441 - recall: 0.8897 - val_loss: 0.1651 - val_accuracy:
0.9386 - val_recall: 0.8936
Epoch 16/30
78/78 [==============================] - 36s 458ms/step - loss: 0.1447
- accuracy: 0.9441 - recall: 0.8882 - val_loss: 0.2080 - val_accuracy:
0.9318 - val_recall: 0.8298
Epoch 17/30
78/78 [==============================] - 35s 450ms/step - loss: 0.1481
- accuracy: 0.9461 - recall: 0.8958 - val_loss: 0.1659 - val_accuracy:
0.9386 - val_recall: 0.8723
Epoch 18/30
78/78 [==============================] - 35s 450ms/step - loss: 0.1484
- accuracy: 0.9469 - recall: 0.8943 - val_loss: 0.1863 - val_accuracy:
0.9364 - val_recall: 0.8511
Epoch 19/30
78/78 [==============================] - 35s 445ms/step - loss: 0.1382
- accuracy: 0.9502 - recall: 0.8988 - val_loss: 0.1948 - val_accuracy:
0.9318 - val_recall: 0.9433
Epoch 20/30
78/78 [==============================] - 34s 442ms/step - loss: 0.1354
- accuracy: 0.9494 - recall: 0.9079 - val_loss: 0.2425 - val_accuracy:
0.9159 - val_recall: 0.7730
Epoch 21/30
78/78 [==============================] - 35s 447ms/step - loss: 0.1285
- accuracy: 0.9538 - recall: 0.8988 - val_loss: 0.2243 - val_accuracy:
0.9159 - val_recall: 0.9504
Epoch 22/30
78/78 [==============================] - 34s 442ms/step - loss: 0.1294
- accuracy: 0.9538 - recall: 0.9169 - val_loss: 0.1636 - val_accuracy:
0.9386 - val_recall: 0.8652
Epoch 23/30
78/78 [==============================] - 35s 443ms/step - loss: 0.1311
- accuracy: 0.9526 - recall: 0.9048 - val_loss: 0.1620 - val_accuracy:
0.9386 - val_recall: 0.9291
Epoch 24/30
78/78 [==============================] - 35s 443ms/step - loss: 0.1228
- accuracy: 0.9554 - recall: 0.9063 - val_loss: 0.1880 - val_accuracy:
0.9295 - val_recall: 0.8723
Epoch 25/30
78/78 [==============================] - 35s 444ms/step - loss: 0.1192
- accuracy: 0.9558 - recall: 0.9169 - val_loss: 0.2074 - val_accuracy:
0.9273 - val_recall: 0.8085
Epoch 26/30
78/78 [==============================] - 35s 444ms/step - loss: 0.1183
- accuracy: 0.9582 - recall: 0.9169 - val_loss: 0.1588 - val_accuracy:
0.9409 - val_recall: 0.8723
Epoch 27/30
78/78 [==============================] - 34s 439ms/step - loss: 0.1209
- accuracy: 0.9506 - recall: 0.8958 - val_loss: 0.1837 - val_accuracy:
0.9318 - val_recall: 0.8582
Epoch 28/30
78/78 [==============================] - 35s 445ms/step - loss: 0.1159
- accuracy: 0.9594 - recall: 0.9169 - val_loss: 0.1970 - val_accuracy:
0.9295 - val_recall: 0.8156
Epoch 29/30
78/78 [==============================] - 34s 436ms/step - loss: 0.1122
```

```
- accuracy: 0.9586 - recall: 0.9199 - val_loss: 0.1654 - val_accuracy:
0.9409 - val_recall: 0.9362
Epoch 30/30
78/78 [==============================] - 35s 452ms/step - loss: 0.1119
- accuracy: 0.9598 - recall: 0.9184 - val_loss: 0.1866 - val_accuracy:
0.9341 - val_recall: 0.8511
```

In [ ]:
```
1 conv_results_train_2 = model_cnn_2.evaluate(X_train_2, y_train_2)
```

```
78/78 [==============================] - 7s 93ms/step - loss: 0.1010 -
accuracy: 0.9658 - recall: 0.8988
```

In [ ]:
```
1 conv_results_test_2 = model_cnn_2.evaluate(X_test_2, y_test_2)
```

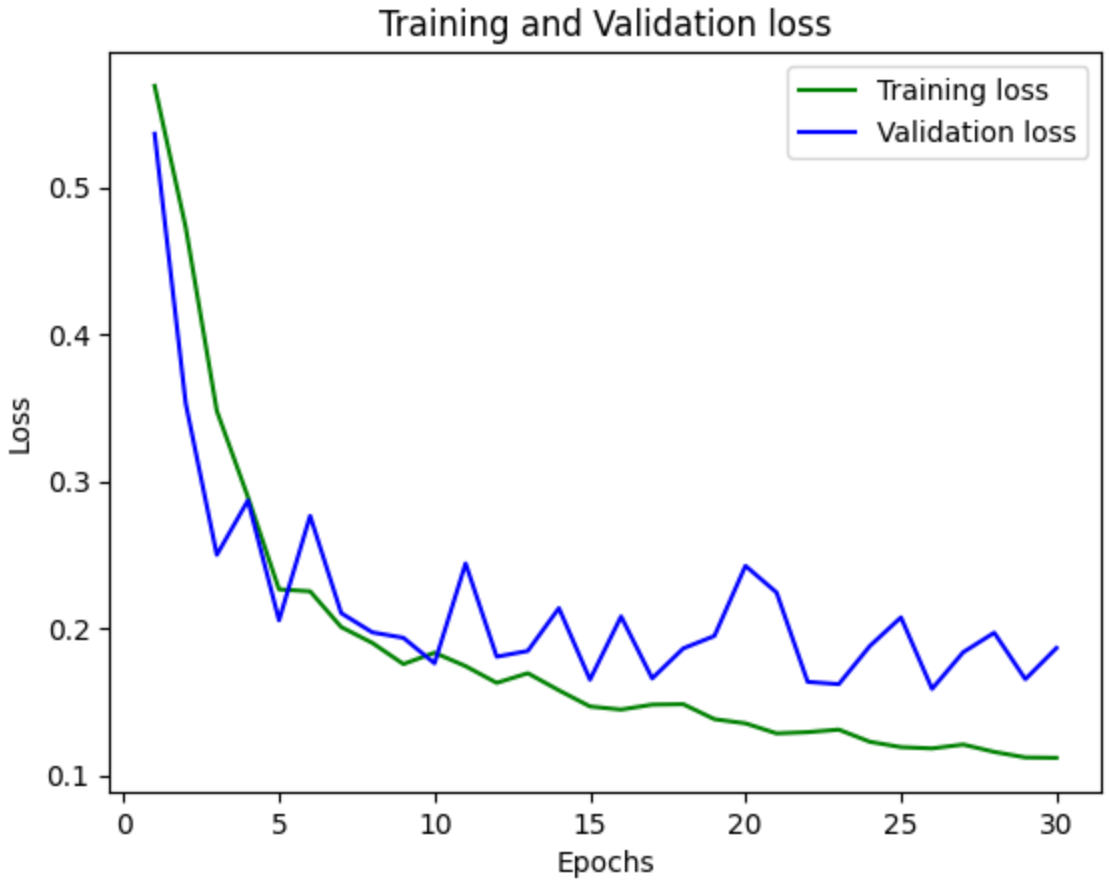
```
14/14 [==============================] - 2s 89ms/step - loss: 0.1866 -
accuracy: 0.9341 - recall: 0.8511
```

In [ ]:
```
1 #model_cnn_2.save('Models/model_cnn_2.keras')
```

In [ ]:
```
1 model_cnn_2 = models.load_model('Models/model_cnn_2.keras')
```

In [ ]:

```
cnn_model_2_val_dict = cnn_history_2.history
loss_values2 = cnn_model_2_val_dict["loss"]
val_loss_values2= cnn_model_2_val_dict["val_loss"]

epochs = range(1,len(loss_values2) + 1)
plt.plot(epochs, loss_values2, "g", label="Training loss")
plt.plot(epochs, val_loss_values2, "blue", label="Validation loss")

plt.title("Training and Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

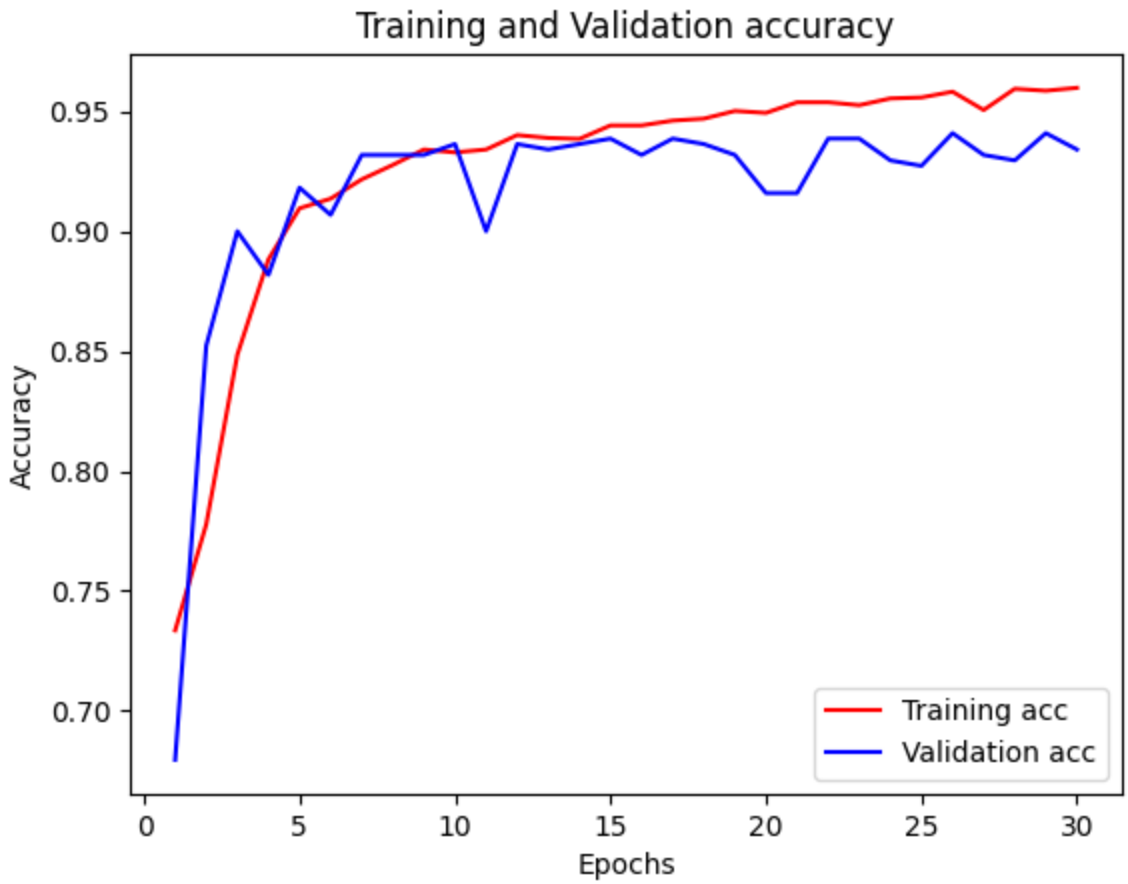


Interestingly, increasing the dense layer and adding two convolution layers decreased the accuracy of the model on test data.

The training accuracy was 96% yet the testing accuracy was 93% and the recall was 85%. The recall significantly dropped from the previous model indicating that the model is not identifying as many pneumonia lungs.

In [ ]:

```
acc_values2 = cnn_model_2_val_dict["accuracy"]
val_acc_values2 = cnn_model_2_val_dict["val_accuracy"]

plt.plot(epochs, acc_values2, "r", label="Training acc")
plt.plot(epochs, val_acc_values2, "blue", label="Validation acc")

plt.title("Training and Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [ ]:
```
Recall_values2 = cnn_model_2_val_dict["recall"]
val_Recall_values2 = cnn_model_2_val_dict["val_recall"]

plt.plot(epochs, Recall_values2, "r", label="Training Recall")
plt.plot(epochs, val_Recall_values2, "blue", label="Validation acc")

plt.title("Training and Validation Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.legend()
plt.show()
```

Third Convolution Neural Network

- Three Convolution layers
- Three Pooling layers
- 30 neurons in the dense layer
- Added a dropout parameter of 0.25

Adding a dropout rate did not significantly change the accuracy off the models. The training accuracy was 94% and the testing accuracy was 95%.

In [ ]:

```
# First Convolutional Layer

model_cnn_3 = models.Sequential()
model_cnn_3.add(layers.Conv2D(32, (3, 3), activation='relu', input_s
model_cnn_3.add(layers.MaxPooling2D((2, 2)))
model_cnn_3.add(layers.Dropout(0.25))

# Second Convolutional Layer

model_cnn_3.add(layers.Conv2D(8, (4, 4), activation='relu',
                        input_shape=(256 ,256,  3)))
model_cnn_3.add(layers.MaxPooling2D((2, 2)))
model_cnn_3.add(layers.Dropout(0.25))


#Third Convolutional Layer

model_cnn_3.add(layers.Conv2D(8, (4, 4), activation='relu', input_sh
model_cnn_3.add(layers.MaxPooling2D((2, 2)))
model_cnn_3.add(layers.Dropout(0.25))


model_cnn_3.add(layers.Flatten())
model_cnn_3.add(layers.Dense(30, activation='relu'))
model_cnn_3.add(layers.Dropout(0.25))
model_cnn_3.add(layers.Dense(1, activation='sigmoid'))

model_cnn_3.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['acc','Recall'])


```

In [ ]:

```
cnn_history_3 = model_cnn_3.fit(X_train_2,
                    y_train_2,
                    epochs=30,
                    batch_size=32,
                    validation_data=(X_test_2, y_test_2))
```

```
Epoch 1/30
78/78 [==============================] - 64s 805ms/step - loss: 0.5769
- acc: 0.7323 - recall: 0.0408 - val_loss: 0.5815 - val_acc: 0.6795 - v
al_recall: 0.0000e+00
Epoch 2/30
78/78 [==============================] - 63s 803ms/step - loss: 0.4788
- acc: 0.7705 - recall: 0.2704 - val_loss: 0.5093 - val_acc: 0.8523 - v
al_recall: 0.8936
Epoch 3/30
78/78 [==============================] - 63s 808ms/step - loss: 0.3974
- acc: 0.8171 - recall: 0.5302 - val_loss: 0.4316 - val_acc: 0.8523 - v
al_recall: 0.9149
Epoch 4/30
78/78 [==============================] - 63s 813ms/step - loss: 0.3332
- acc: 0.8561 - recall: 0.6601 - val_loss: 0.2959 - val_acc: 0.8955 - v
al_recall: 0.7730
Epoch 5/30
78/78 [==============================] - 63s 811ms/step - loss: 0.3017
- acc: 0.8770 - recall: 0.7205 - val_loss: 0.2770 - val_acc: 0.9114 - v
al_recall: 0.8440
Epoch 6/30
78/78 [==============================] - 63s 814ms/step - loss: 0.2814
- acc: 0.8826 - recall: 0.7492 - val_loss: 0.2535 - val_acc: 0.9182 - v
al_recall: 0.8511
Epoch 7/30
78/78 [==============================] - 64s 819ms/step - loss: 0.2592
- acc: 0.8955 - recall: 0.7734 - val_loss: 0.2918 - val_acc: 0.9159 - v
al_recall: 0.9149
Epoch 8/30
78/78 [==============================] - 64s 823ms/step - loss: 0.2504
- acc: 0.8999 - recall: 0.7915 - val_loss: 0.2251 - val_acc: 0.9295 - v
al_recall: 0.8652
Epoch 9/30
78/78 [==============================] - 64s 817ms/step - loss: 0.2568
- acc: 0.9019 - recall: 0.7991 - val_loss: 0.2306 - val_acc: 0.9250 - v
al_recall: 0.9078
Epoch 10/30
78/78 [==============================] - 66s 851ms/step - loss: 0.2307
- acc: 0.9140 - recall: 0.8233 - val_loss: 0.2102 - val_acc: 0.9205 - v
al_recall: 0.8582
Epoch 11/30
78/78 [==============================] - 65s 840ms/step - loss: 0.2369
- acc: 0.9152 - recall: 0.8278 - val_loss: 0.3015 - val_acc: 0.8977 - v
al_recall: 0.9291
Epoch 12/30
78/78 [==============================] - 66s 847ms/step - loss: 0.2163
- acc: 0.9180 - recall: 0.8263 - val_loss: 0.3477 - val_acc: 0.8659 - v
al_recall: 0.9574
Epoch 13/30
78/78 [==============================] - 64s 824ms/step - loss: 0.2291
- acc: 0.9140 - recall: 0.8278 - val_loss: 0.2354 - val_acc: 0.9159 - v
al_recall: 0.9220
Epoch 14/30
78/78 [==============================] - 65s 828ms/step - loss: 0.2113
- acc: 0.9220 - recall: 0.8414 - val_loss: 0.3238 - val_acc: 0.8773 - v
al_recall: 0.9574
Epoch 15/30
```

```
78/78 [==============================] - 67s 854ms/step - loss: 0.2207
- acc: 0.9168 - recall: 0.8278 - val_loss: 0.3045 - val_acc: 0.9136 - v
al_recall: 0.9433
Epoch 16/30
78/78 [==============================] - 64s 818ms/step - loss: 0.2137
- acc: 0.9184 - recall: 0.8399 - val_loss: 0.2974 - val_acc: 0.9045 - v
al_recall: 0.9504
Epoch 17/30
78/78 [==============================] - 64s 821ms/step - loss: 0.2115
- acc: 0.9200 - recall: 0.8429 - val_loss: 0.2563 - val_acc: 0.9295 - v
al_recall: 0.9362
Epoch 18/30
78/78 [==============================] - 72s 921ms/step - loss: 0.1810
- acc: 0.9309 - recall: 0.8686 - val_loss: 0.2047 - val_acc: 0.9295 - v
al_recall: 0.9220
Epoch 19/30
78/78 [==============================] - 68s 870ms/step - loss: 0.1939
- acc: 0.9248 - recall: 0.8565 - val_loss: 0.2091 - val_acc: 0.9364 - v
al_recall: 0.8582
Epoch 20/30
78/78 [==============================] - 66s 843ms/step - loss: 0.1939
- acc: 0.9216 - recall: 0.8369 - val_loss: 0.1929 - val_acc: 0.9364 - v
al_recall: 0.8794
Epoch 21/30
78/78 [==============================] - 65s 831ms/step - loss: 0.1824
- acc: 0.9297 - recall: 0.8656 - val_loss: 0.1923 - val_acc: 0.9386 - v
al_recall: 0.8652
Epoch 22/30
78/78 [==============================] - 66s 848ms/step - loss: 0.1820
- acc: 0.9341 - recall: 0.8716 - val_loss: 0.1972 - val_acc: 0.9409 - v
al_recall: 0.8865
Epoch 23/30
78/78 [==============================] - 66s 840ms/step - loss: 0.1801
- acc: 0.9397 - recall: 0.8731 - val_loss: 0.1921 - val_acc: 0.9295 - v
al_recall: 0.9149
Epoch 24/30
78/78 [==============================] - 65s 833ms/step - loss: 0.1720
- acc: 0.9317 - recall: 0.8610 - val_loss: 0.1922 - val_acc: 0.9318 - v
al_recall: 0.9078
Epoch 25/30
78/78 [==============================] - 65s 834ms/step - loss: 0.1705
- acc: 0.9377 - recall: 0.8671 - val_loss: 0.1837 - val_acc: 0.9295 - v
al_recall: 0.8936
Epoch 26/30
78/78 [==============================] - 65s 831ms/step - loss: 0.1753
- acc: 0.9329 - recall: 0.8746 - val_loss: 0.1888 - val_acc: 0.9386 - v
al_recall: 0.9007
Epoch 27/30
78/78 [==============================] - 67s 864ms/step - loss: 0.1729
- acc: 0.9345 - recall: 0.8761 - val_loss: 0.1868 - val_acc: 0.9227 - v
al_recall: 0.8227
Epoch 28/30
78/78 [==============================] - 65s 834ms/step - loss: 0.1616
- acc: 0.9369 - recall: 0.8792 - val_loss: 0.1853 - val_acc: 0.9295 - v
al_recall: 0.9078
Epoch 29/30
78/78 [==============================] - 65s 840ms/step - loss: 0.1699
```

```
- acc: 0.9357 - recall: 0.8731 - val_loss: 0.1680 - val_acc: 0.9432 - v
al_recall: 0.9220
Epoch 30/30
78/78 [==============================] - 66s 844ms/step - loss: 0.1572
- acc: 0.9457 - recall: 0.8927 - val_loss: 0.1806 - val_acc: 0.9386 - v
al_recall: 0.8865
```

In [ ]:
```
1 #model_cnn_3.save('Models/model_cnn_3.keras')
```

In [ ]:
```
1 model_cnn_3 = models.load_model('Models/model_cnn_3.keras')
```

In [ ]:
```
1 conv_results_train_3 = model_cnn_3.evaluate(X_train_2, y_train_2)
```

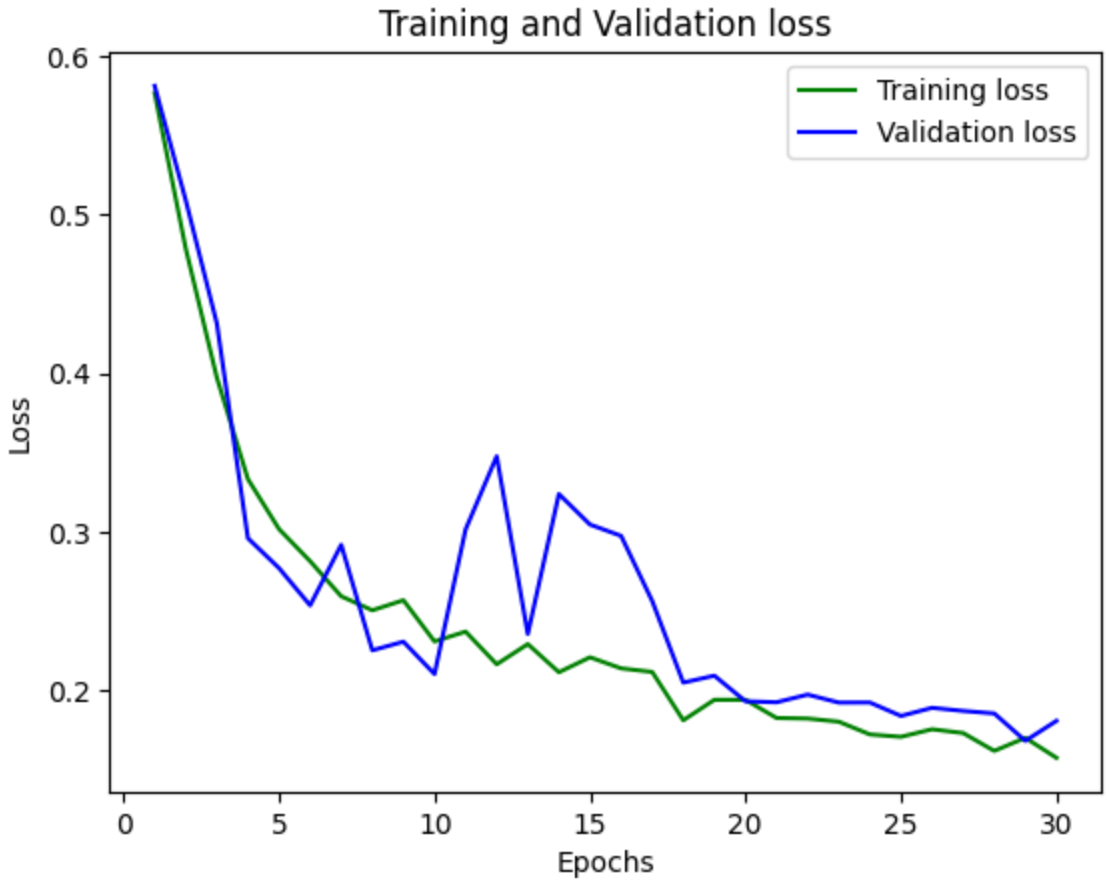
```
78/78 [==============================] - 13s 169ms/step - loss: 0.1558
- acc: 0.9542 - recall: 0.9018
```

In [ ]:
```
1 conv_results_test_3 = model_cnn_3.evaluate(X_test_2, y_test_2)
```

```
14/14 [==============================] - 2s 157ms/step - loss: 0.1806 -
acc: 0.9386 - recall: 0.8865
```

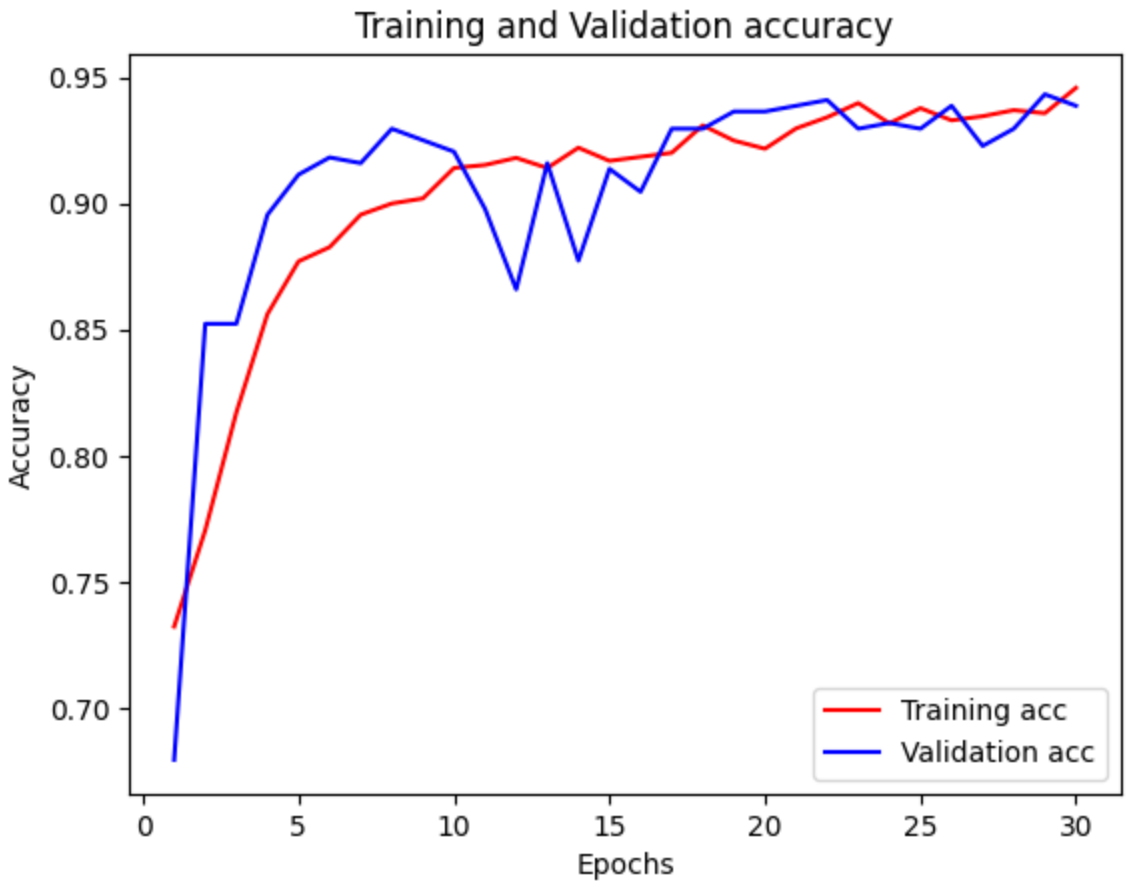
In [ ]:

```
cnn_model_3_val_dict = cnn_history_3.history
loss_values3 = cnn_model_3_val_dict["loss"]
val_loss_values3= cnn_model_3_val_dict["val_loss"]

epochs = range(1,len(loss_values3) + 1)
plt.plot(epochs, loss_values3, "g", label="Training loss")
plt.plot(epochs, val_loss_values3, "blue", label="Validation loss")

plt.title("Training and Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

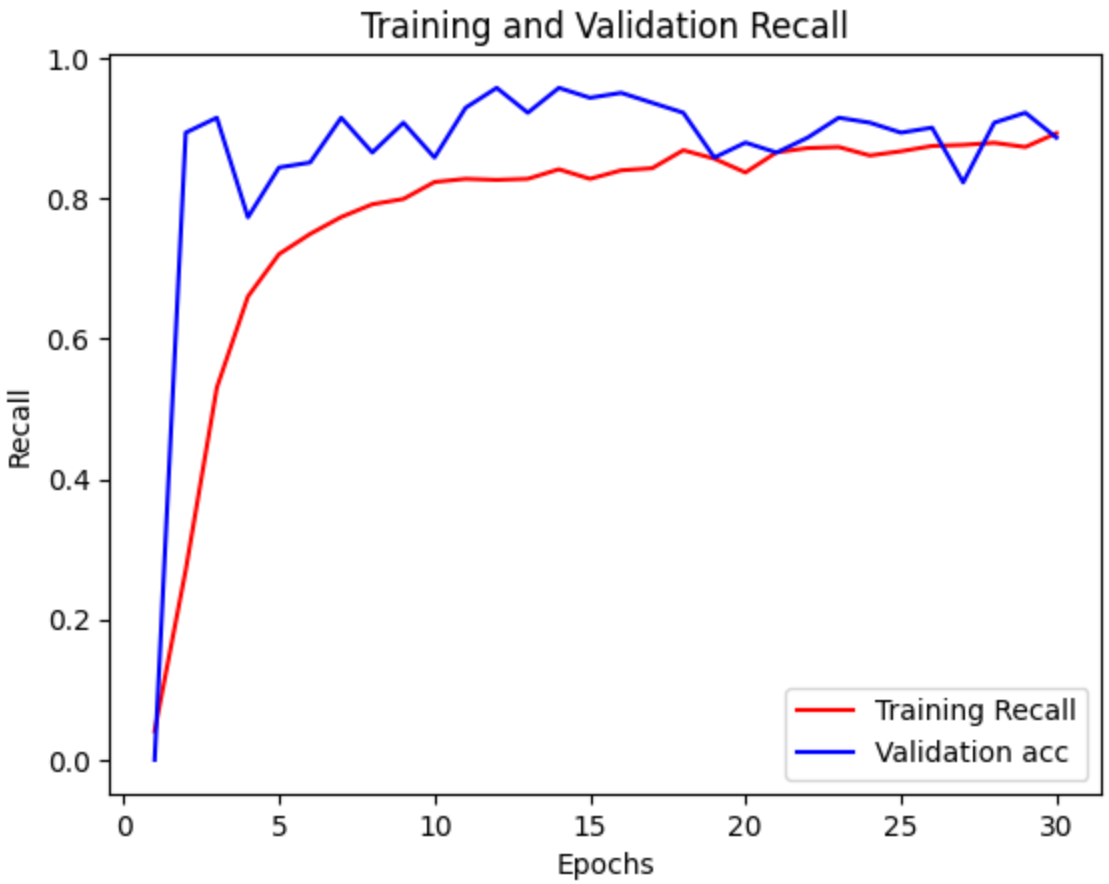


In [ ]:

```
acc_values3 = cnn_model_3_val_dict["acc"]
val_acc_values3 = cnn_model_3_val_dict["val_acc"]

plt.plot(epochs, acc_values3, "r", label="Training acc")
plt.plot(epochs, val_acc_values3, "blue", label="Validation acc")

plt.title("Training and Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [ ]:
```
Recall_values3 = cnn_model_3_val_dict["recall"]
val_Recall_values3 = cnn_model_3_val_dict["val_recall"]

plt.plot(epochs, Recall_values3, "r", label="Training Recall")
plt.plot(epochs, val_Recall_values3, "blue", label="Validation acc")

plt.title("Training and Validation Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.legend()
plt.show()
```



Adding a dropout rate did not significantly change the accuracy off the models. The training accuracy was 94% and the testing accuracy was 95%. The recall was 89%. Slightly higher than the past two iterations.

Fourth Convolution Neural Network

- Three Convolution layers
- Three Pooling layers
- 30 neurons in the dense layer
- Added a dropout parameter of 0.25
- L2 regularizer off 0.001 added

In [ ]:

```
# First Convolutional Layer

model_cnn_4 = models.Sequential()
model_cnn_4.add(layers.Conv2D(32, (3, 3), activation='relu',
                         kernel_regularizer=keras.regularizers.l2(l=0
model_cnn_4.add(layers.MaxPooling2D((2, 2)))
model_cnn_4.add(layers.Dropout(0.25))

# Second Convolutional Layer

model_cnn_4.add(layers.Conv2D(8, (4, 4), activation='relu',
                              kernel_regularizer=keras.regularizers.
                         input_shape=(256 ,256,  3)))
model_cnn_4.add(layers.MaxPooling2D((2, 2)))
model_cnn_4.add(layers.Dropout(0.25))


#Third Convolutional Layer

model_cnn_4.add(layers.Conv2D(8, (4, 4), activation='relu',
                              kernel_regularizer=keras.regularizers.
                              input_shape=(256 ,256,  3)))
model_cnn_4.add(layers.MaxPooling2D((2, 2)))
model_cnn_4.add(layers.Dropout(0.25))

model_cnn_4.add(layers.Flatten())
model_cnn_4.add(layers.Dense(30, activation='relu'))
model_cnn_4.add(layers.Dropout(0.25))
model_cnn_4.add(layers.Dense(1, activation='sigmoid'))

model_cnn_4.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['acc','Recall'])


```

In [ ]:

```
cnn_history_4 = model_cnn_4.fit(X_train_2,
                    y_train_2,
                    epochs=30,
                    batch_size=32,
                    validation_data=(X_test_2, y_test_2))
```

```
Epoch 1/30
78/78 [==============================] - 69s 875ms/step - loss: 0.6038
- acc: 0.7251 - recall: 0.0076 - val_loss: 0.6269 - val_acc: 0.6886 - v
al_recall: 0.0284
Epoch 2/30
78/78 [==============================] - 68s 868ms/step - loss: 0.5050
- acc: 0.7661 - recall: 0.2870 - val_loss: 0.4569 - val_acc: 0.7795 - v
al_recall: 0.3121
Epoch 3/30
78/78 [==============================] - 68s 873ms/step - loss: 0.4290
- acc: 0.8236 - recall: 0.5725 - val_loss: 0.3932 - val_acc: 0.8477 - v
al_recall: 0.5603
Epoch 4/30
78/78 [==============================] - 73s 935ms/step - loss: 0.3619
- acc: 0.8589 - recall: 0.6692 - val_loss: 0.3662 - val_acc: 0.8932 - v
al_recall: 0.8794
Epoch 5/30
78/78 [==============================] - 68s 867ms/step - loss: 0.3123
- acc: 0.8778 - recall: 0.7417 - val_loss: 0.2965 - val_acc: 0.9068 - v
al_recall: 0.8369
Epoch 6/30
78/78 [==============================] - 68s 867ms/step - loss: 0.2992
- acc: 0.8899 - recall: 0.7644 - val_loss: 0.3258 - val_acc: 0.8795 - v
al_recall: 0.9291
Epoch 7/30
78/78 [==============================] - 71s 908ms/step - loss: 0.2777
- acc: 0.8983 - recall: 0.8006 - val_loss: 0.2478 - val_acc: 0.9182 - v
al_recall: 0.8794
Epoch 8/30
78/78 [==============================] - 69s 891ms/step - loss: 0.2644
- acc: 0.9068 - recall: 0.8157 - val_loss: 0.2756 - val_acc: 0.9045 - v
al_recall: 0.9220
Epoch 9/30
78/78 [==============================] - 68s 876ms/step - loss: 0.2658
- acc: 0.9080 - recall: 0.8112 - val_loss: 0.2634 - val_acc: 0.9227 - v
al_recall: 0.9007
Epoch 10/30
78/78 [==============================] - 68s 870ms/step - loss: 0.2584
- acc: 0.9051 - recall: 0.8142 - val_loss: 0.2908 - val_acc: 0.9091 - v
al_recall: 0.9220
Epoch 11/30
78/78 [==============================] - 69s 884ms/step - loss: 0.2437
- acc: 0.9144 - recall: 0.8278 - val_loss: 0.2953 - val_acc: 0.8864 - v
al_recall: 0.9291
Epoch 12/30
78/78 [==============================] - 68s 874ms/step - loss: 0.2471
- acc: 0.9140 - recall: 0.8278 - val_loss: 0.2910 - val_acc: 0.8955 - v
al_recall: 0.9220
Epoch 13/30
78/78 [==============================] - 69s 882ms/step - loss: 0.2395
- acc: 0.9104 - recall: 0.8278 - val_loss: 0.2492 - val_acc: 0.9250 - v
al_recall: 0.8936
Epoch 14/30
78/78 [==============================] - 70s 902ms/step - loss: 0.2321
- acc: 0.9196 - recall: 0.8278 - val_loss: 0.2374 - val_acc: 0.9273 - v
al_recall: 0.9078
Epoch 15/30
```

```
78/78 [==============================] - 69s 884ms/step - loss: 0.2229
- acc: 0.9244 - recall: 0.8520 - val_loss: 0.2537 - val_acc: 0.9159 - v
al_recall: 0.9220
Epoch 16/30
78/78 [==============================] - 69s 882ms/step - loss: 0.2285
- acc: 0.9244 - recall: 0.8610 - val_loss: 0.2402 - val_acc: 0.9386 - v
al_recall: 0.8794
Epoch 17/30
78/78 [==============================] - 68s 866ms/step - loss: 0.2224
- acc: 0.9212 - recall: 0.8444 - val_loss: 0.3173 - val_acc: 0.9023 - v
al_recall: 0.9574
Epoch 18/30
78/78 [==============================] - 68s 873ms/step - loss: 0.2215
- acc: 0.9252 - recall: 0.8474 - val_loss: 0.2695 - val_acc: 0.8932 - v
al_recall: 0.9433
Epoch 19/30
78/78 [==============================] - 67s 863ms/step - loss: 0.2117
- acc: 0.9321 - recall: 0.8671 - val_loss: 0.2530 - val_acc: 0.9295 - v
al_recall: 0.9220
Epoch 20/30
78/78 [==============================] - 67s 862ms/step - loss: 0.2067
- acc: 0.9337 - recall: 0.8716 - val_loss: 0.2103 - val_acc: 0.9364 - v
al_recall: 0.9078
Epoch 21/30
78/78 [==============================] - 68s 877ms/step - loss: 0.2071
- acc: 0.9289 - recall: 0.8550 - val_loss: 0.2308 - val_acc: 0.9250 - v
al_recall: 0.9362
Epoch 22/30
78/78 [==============================] - 69s 881ms/step - loss: 0.1970
- acc: 0.9361 - recall: 0.8701 - val_loss: 0.2183 - val_acc: 0.9295 - v
al_recall: 0.8440
Epoch 23/30
78/78 [==============================] - 68s 872ms/step - loss: 0.1936
- acc: 0.9381 - recall: 0.8822 - val_loss: 0.1972 - val_acc: 0.9386 - v
al_recall: 0.8936
Epoch 24/30
78/78 [==============================] - 69s 885ms/step - loss: 0.1956
- acc: 0.9397 - recall: 0.8792 - val_loss: 0.2159 - val_acc: 0.9250 - v
al_recall: 0.9220
Epoch 25/30
78/78 [==============================] - 68s 873ms/step - loss: 0.1937
- acc: 0.9345 - recall: 0.8671 - val_loss: 0.2209 - val_acc: 0.9250 - v
al_recall: 0.9291
Epoch 26/30
78/78 [==============================] - 68s 871ms/step - loss: 0.1891
- acc: 0.9389 - recall: 0.8776 - val_loss: 0.2050 - val_acc: 0.9295 - v
al_recall: 0.9362
Epoch 27/30
78/78 [==============================] - 69s 886ms/step - loss: 0.1889
- acc: 0.9345 - recall: 0.8761 - val_loss: 0.2005 - val_acc: 0.9386 - v
al_recall: 0.9149
Epoch 28/30
78/78 [==============================] - 70s 893ms/step - loss: 0.1873
- acc: 0.9377 - recall: 0.8852 - val_loss: 0.2098 - val_acc: 0.9409 - v
al_recall: 0.9007
Epoch 29/30
78/78 [==============================] - 69s 885ms/step - loss: 0.1901
```

```
- acc: 0.9405 - recall: 0.8852 - val_loss: 0.2112 - val_acc: 0.9386 - v
al_recall: 0.9362
Epoch 30/30
78/78 [==============================] - 69s 890ms/step - loss: 0.1815
- acc: 0.9457 - recall: 0.8912 - val_loss: 0.2002 - val_acc: 0.9295 - v
al_recall: 0.9149
```

In [ ]:
```
#model_cnn_4.save('Models/model_cnn_4_recall.keras')
```

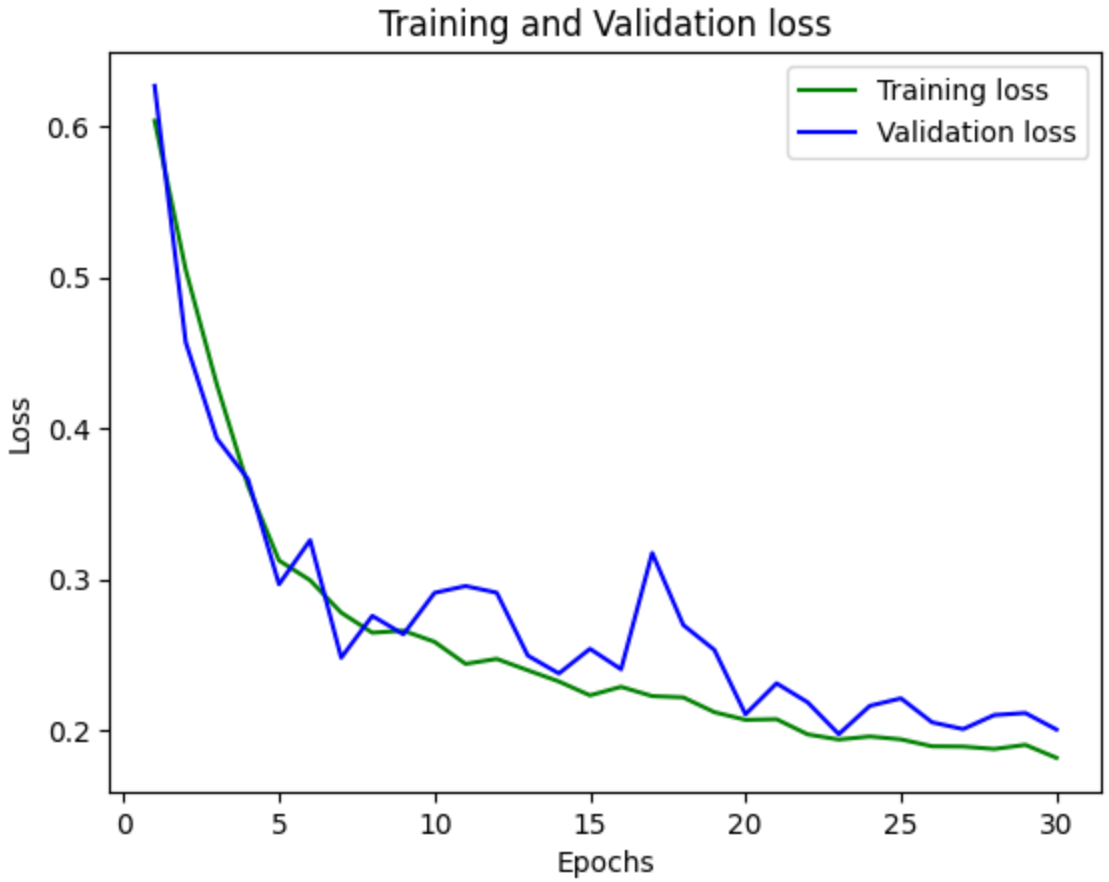
In [ ]:
```
#model_cnn_4 = models.load_model('Models/model_cnn_4.keras')
```

In [ ]:
```
conv_results_test_4 = model_cnn_4.evaluate(X_test_2, y_test_2)
```

```
14/14 [==============================] - 2s 173ms/step - loss: 0.2002 -
acc: 0.9295 - recall: 0.9149
```

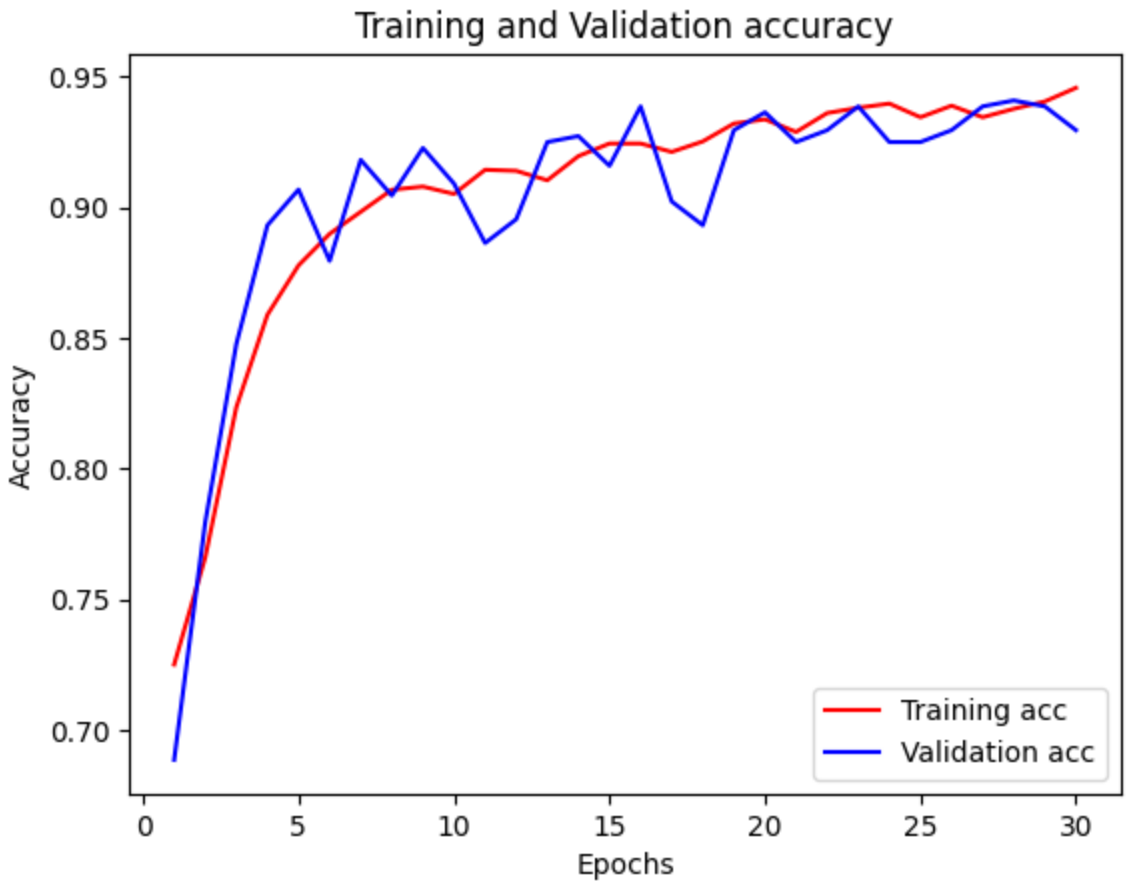
In [ ]:

```
cnn_model_4_val_dict = cnn_history_4.history
loss_values4 = cnn_model_4_val_dict["loss"]
val_loss_values4= cnn_model_4_val_dict["val_loss"]

epochs = range(1,len(loss_values4) + 1)
plt.plot(epochs, loss_values4, "g", label="Training loss")
plt.plot(epochs, val_loss_values4, "blue", label="Validation loss")

plt.title("Training and Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

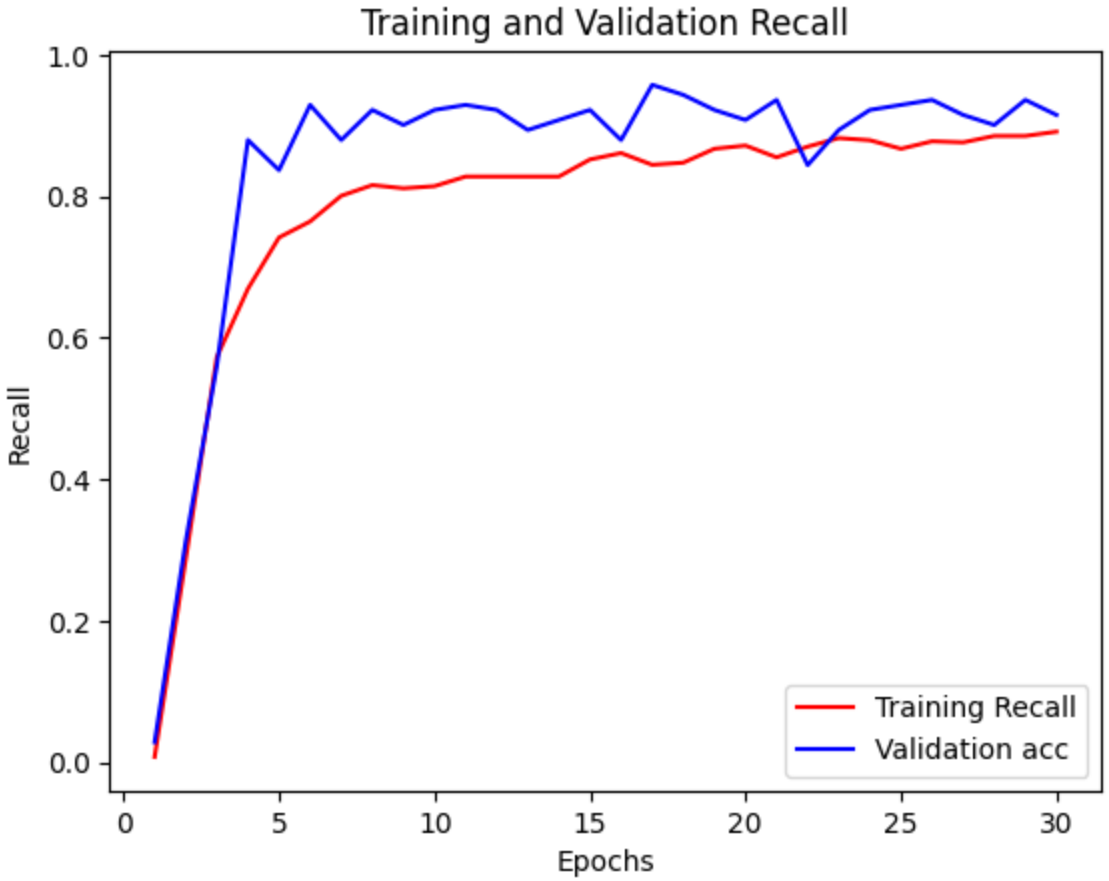


In [ ]:

```
acc_values4 = cnn_model_4_val_dict["acc"]
val_acc_values4 = cnn_model_4_val_dict["val_acc"]

plt.plot(epochs, acc_values4, "r", label="Training acc")
plt.plot(epochs, val_acc_values4, "blue", label="Validation acc")

plt.title("Training and Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [ ]:
```
Recall_values4 = cnn_model_4_val_dict["recall"]
val_Recall_values4 = cnn_model_4_val_dict["val_recall"]

plt.plot(epochs, Recall_values4, "r", label="Training Recall")
plt.plot(epochs, val_Recall_values4, "blue", label="Validation acc")

plt.title("Training and Validation Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.legend()
plt.show()
```



Model Evaluation

We've run 4 models. All have had very good accuracy (over 93%) on both the training and testing data. Let's run these models over the the larger testing data set that was split off earlier.

In [ ]:
```
# Run CNN #1

CNN_Results_1 = model_cnn_1.evaluate(X_test_1,y_test_1)

CNN_Results_1
```

```
92/92 [==============================] - 7s 71ms/step - loss: 0.1265 -
accuracy: 0.9549 - recall: 0.8923
```

Out[36]:
```
[0.12648145854473114, 0.9549180269241333, 0.892307698726654]
```

In [ ]:
```
# Run CNN #2

CNN_Results_2 = model_cnn_2.evaluate(X_test_1,y_test_1)

CNN_Results_2
```

```
92/92 [==============================] - 9s 92ms/step - loss: 0.1445 -
accuracy: 0.9491 - recall: 0.8577
```

Out[37]:
```
[0.1444677859544754, 0.9491119980812073, 0.857692301273346]
```

In [ ]:
```
# Run CNN #3 Model

CNN_Results_3 = model_cnn_3.evaluate(X_test_1,y_test_1)

CNN_Results_3

```

```
92/92 [==============================] - 16s 169ms/step - loss: 0.1595
- acc: 0.9529 - recall: 0.8846
```

Out[38]:
```
[0.1594732701778412, 0.9528688788414001, 0.8846153616905212]
```

In [ ]:
```
#Run CNN #4 Model

CNN_Results_4 = model_cnn_4.evaluate(X_test_1,y_test_1)

CNN_Results_4

```

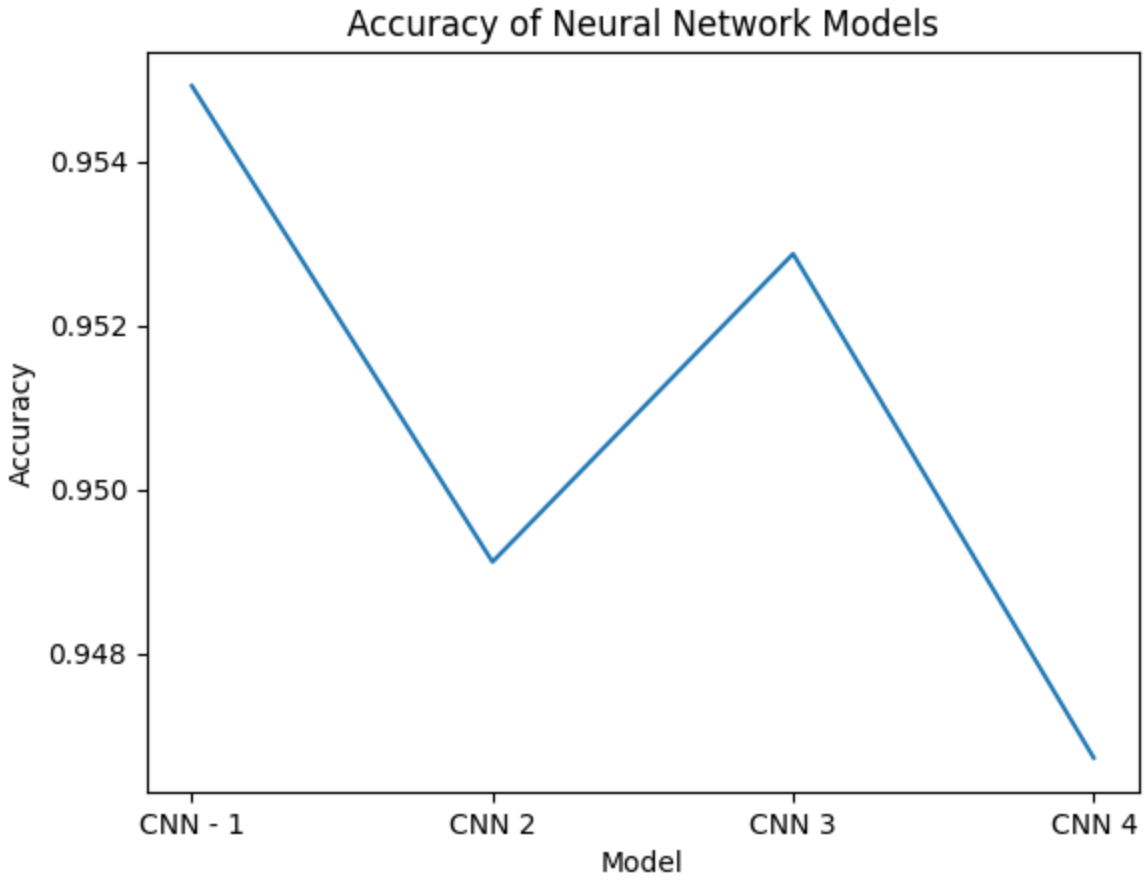
```
92/92 [==============================] - 16s 173ms/step - loss: 0.1806
- acc: 0.9467 - recall: 0.9231
```

Out[39]:
```
[0.1806493103504181, 0.9467213153839111, 0.9230769276618958]
```

In [ ]:
```
Models_Accuracy = [CNN_Results_1[1],CNN_Results_2[1],CNN_Results_3[1

Models_Accuracy
```

Out[40]:
```
[0.9549180269241333,
 0.9491119980812073,
 0.9528688788414001,
 0.9467213153839111]
```

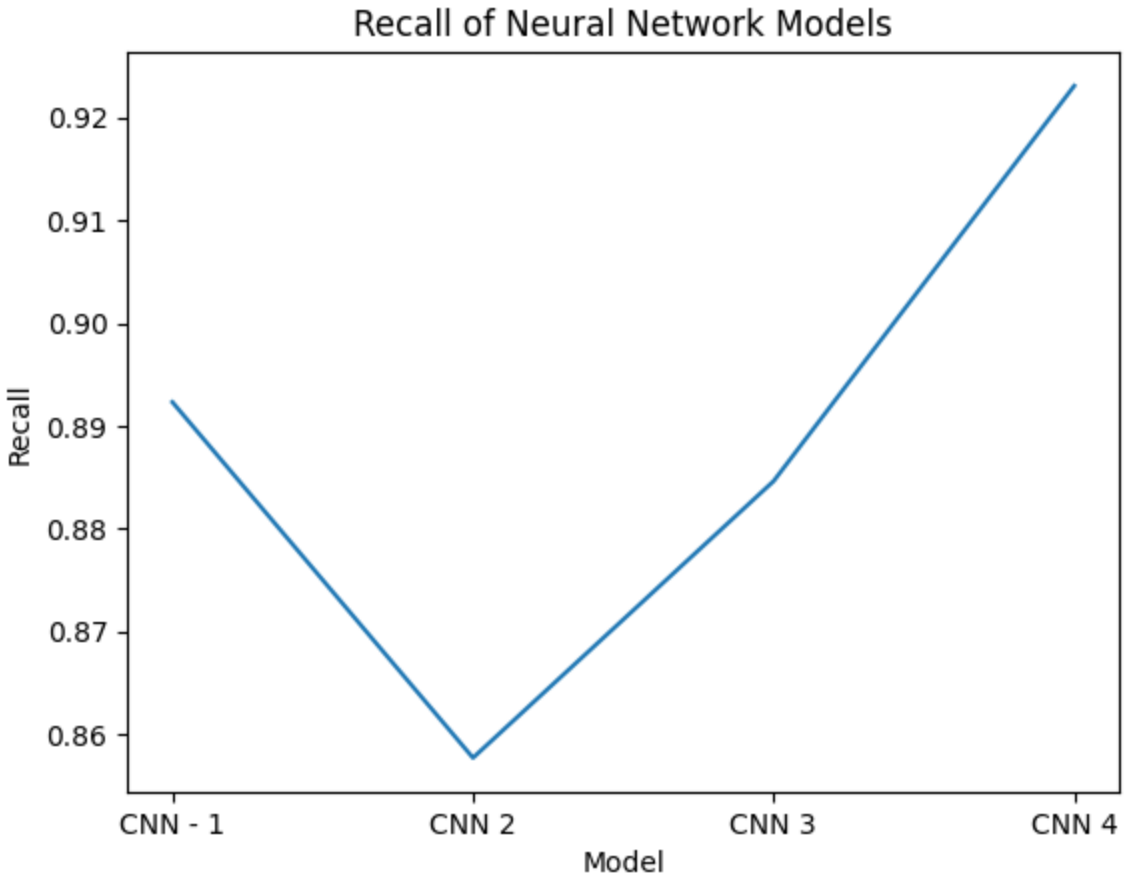
In [ ]:
```
sns.lineplot(x = ["CNN - 1","CNN 2","CNN 3", "CNN 4"],y = Models_Acc

plt.title("Accuracy of Neural Network Models")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.show()
```



In [ ]:
```
# Do the same for recall

Models_Recall = [CNN_Results_1[2],CNN_Results_2[2],CNN_Results_3[2],

Models_Recall
```

Out[44]:
```
[0.892307698726654, 0.857692301273346, 0.8846153616905212, 0.9230769276
618958]
```

In [ ]:
```
sns.lineplot(x = ["CNN - 1","CNN 2","CNN 3", "CNN 4"],y = Models_Rec

plt.title("Recall of Neural Network Models")
plt.xlabel("Model")
plt.ylabel("Recall")
plt.show()
```



Review of neural networks implemented above

- Base Neural Netork: No convolution. 3 layers.
- CNN 1: One Convolution layer
- CNN 2: Three Convolution Layers
- CNN 3: Dropout of 0.25 added
- CNN 4: Regularizer added

Final Model Evaluation

Based on the accuracies above, the first neural network with one convolution layers and no regularizers or drop out had the highest accuracy.

Ultimately, all models had accuracies within a percentage point from one - another. It would not be surprising if one of the models with regularization performed better on a different data set with significantly more records.

If this were based on accuracy alone then the first convolution neural network would be the choice.

However, the first, second, and third models had lower recalls. The fourth models had a higher recall by nearly 3-6 percentage points.

In addition, when reviewing the plots off the accuracy/loss over the epochs, we noticed that the models without any regulation had significant variance than the ones that did not.

Given the fact that the fourth model had a higher recall and less variability, the fourth model is the one we choose.

In [ ]:
```
# First get the Last 10 values of the validation accuracy and store

cnn_1_acc_last_10 = cnn_model_1_val_dict["val_accuracy"][20:]

cnn_2_acc_last_10 = cnn_model_2_val_dict["val_accuracy"][20:]

cnn_3_acc_last_10 = cnn_model_3_val_dict["val_acc"][20:]

cnn_4_acc_last_10 = cnn_model_4_val_dict["val_acc"][20:]


# Get the standard deviation off those values

cnn_1_acc_std = np.std(cnn_1_acc_last_10)

cnn_2_acc_std = np.std(cnn_2_acc_last_10)

cnn_3_acc_std = np.std(cnn_3_acc_last_10)

cnn_4_acc_std = np.std(cnn_4_acc_last_10)

```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-43-aa3769f58561> in <cell line: 3>()
      1 # First get the Last 10 values of the validation accuracy and store them in an array
      2
----> 3 cnn_1_acc_last_10 = cnn_model_1_val_dict["val_acc"][20:]
      4
      5 cnn_2_acc_last_10 = cnn_model_2_val_dict["val_accuracy"][20:]

NameError: name 'cnn_model_1_val_dict' is not defined
```

In [ ]:
```
Models_Std_lst_10 = [cnn_1_acc_std,cnn_2_acc_std,cnn_3_acc_std,cnn_4

sns.lineplot(x = ["CNN - 1","CNN 2","CNN 3", "CNN 4"],y = Models_Std

plt.title("Standard Deviation in Validation Accuracy During the Last
plt.xlabel("Model")
plt.ylabel("Validation Accuracy Standard Deviation")
plt.show()
```

Charting the Standard Deviation in Accuracy during the Last 5 Epochs

In [ ]:
```
# First get the Last 5 values of the validation accuracy and store t

cnn_1_acc_last_5 = cnn_model_1_val_dict["val_accuracy"][25:]

cnn_2_acc_last_5 = cnn_model_2_val_dict["val_accuracy"][25:]

cnn_3_acc_last_5 = cnn_model_3_val_dict["val_acc"][25:]

cnn_4_acc_last_5 = cnn_model_4_val_dict["val_acc"][25:]


# Get the standard deviation off those values

cnn_1_acc_std_5 = np.std(cnn_1_acc_last_5)

cnn_2_acc_std_5 = np.std(cnn_2_acc_last_5)

cnn_3_acc_std_5 = np.std(cnn_3_acc_last_5)

cnn_4_acc_std_5 = np.std(cnn_4_acc_last_5)

```

In [ ]:
```
Models_Std_lst_5 = [cnn_1_acc_std_5,cnn_2_acc_std_5,cnn_3_acc_std_5,

sns.lineplot(x = ["CNN - 1","CNN 2","CNN 3", "CNN 4"],y = Models_Std

plt.title("Standard Deviation in Validation Accuracy During the Last
plt.xlabel("Model")
plt.ylabel("Validation Accuracy Standard Deviation")
plt.show()
```

In [ ]:
```
for i in range(0,4):
    print((Models_Std_lst_10[i]-Models_Std_lst_5[i])/(Models_Std_lst
```

```
0.0
0.36271649307452364
-0.18738361104841839
0.6602001802152138
```

The Standard Deviation in the accuracy decreased for CNN models 1. It remained roughly the same in model 3, but increased in Models 2 and 4. Indicating that the models were getting more stable in their predictions. For CNN Model 2, the standard deviation increased instead.

Even though CNN model 2 had the highest accuracy, because it showed a high variance in accuracy during the epochs it will not be the final model selected.

The model selected will be CNN model 3. Model 3 has the next highest accuracy (94.5%) and showed that the predictions were becoming stable as the number of epochs increased. As a result, this model is more likely to show less variability when utilized. This is likely due to the regularization, specifically the dropout parameter, added to this model.

Let's look at some of the parameters in the model.

In [ ]:
```
1 model_cnn_4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 254, 254, 32)      896

 max_pooling2d_7 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 dropout_4 (Dropout)         (None, 127, 127, 32)      0

 conv2d_8 (Conv2D)           (None, 124, 124, 8)       4104

 max_pooling2d_8 (MaxPoolin  (None, 62, 62, 8)         0
 g2D)

 dropout_5 (Dropout)         (None, 62, 62, 8)         0

 conv2d_9 (Conv2D)           (None, 59, 59, 8)         1032

 max_pooling2d_9 (MaxPoolin  (None, 29, 29, 8)         0
 g2D)

 dropout_6 (Dropout)         (None, 29, 29, 8)         0

 flatten_3 (Flatten)         (None, 6728)              0

 dense_6 (Dense)             (None, 30)                201870

 dropout_7 (Dropout)         (None, 30)                0

 dense_7 (Dense)             (None, 1)                 31

=================================================================
Total params: 207933 (812.24 KB)
Trainable params: 207933 (812.24 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The final model is a convolutional neural network model. It consists of 3 convolutional layers, 3 pooling layers, a flattening layer, and two final dense layers. The convolutional layers have filters of size (3,3). The first layer has 32 filters, followed by 8 filters in the second.

Each layer has a dropout for regularization.

Max pooling is done with a (2,2) dimension. The first dense layer contains 30 neurons, and the output layer contains one neuron with a sigmoid activation function. The final model has a 94.5% accuracy.

In [ ]:
```
1 y_prediction_1
```

Out[154]:
```
array([[-1.6602764],
       [-2.0248995],
       [-5.1376843],
       ...,
       [-1.5081309],
       [-5.7306857],
       [-4.535429 ]], dtype=float32)
```

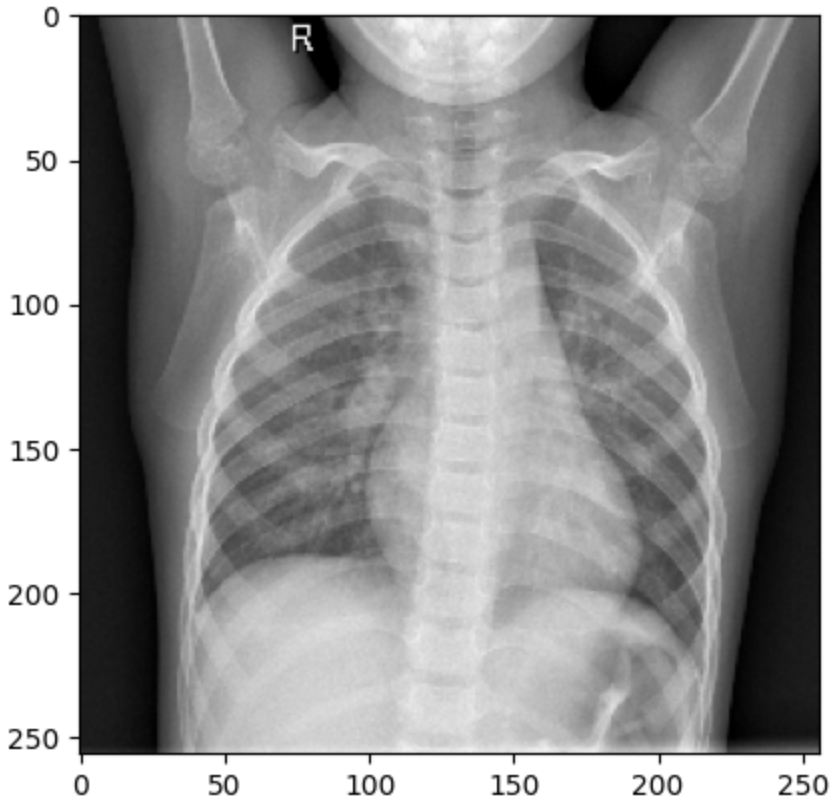
Feature Analysis

Let's look at the features and the parts of the image our model emphasized to determine if a lung was pneumatic. Given the amount of data in an image, we will show images of lungs from the data set and highlight the portions that were activated by the model.

In [ ]:

```
from keras.preprocessing import image

img_path = 'chest_xray/train/NORMAL/IM-0115-0001.jpeg'


img = image.load_img(img_path, target_size=(256, 256))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)

#Follow the Original Model Preprocessing
img_tensor /= 255.

#Check tensor shape
print(img_tensor.shape)

#Preview an image
plt.imshow(img_tensor[0])
plt.show()
```

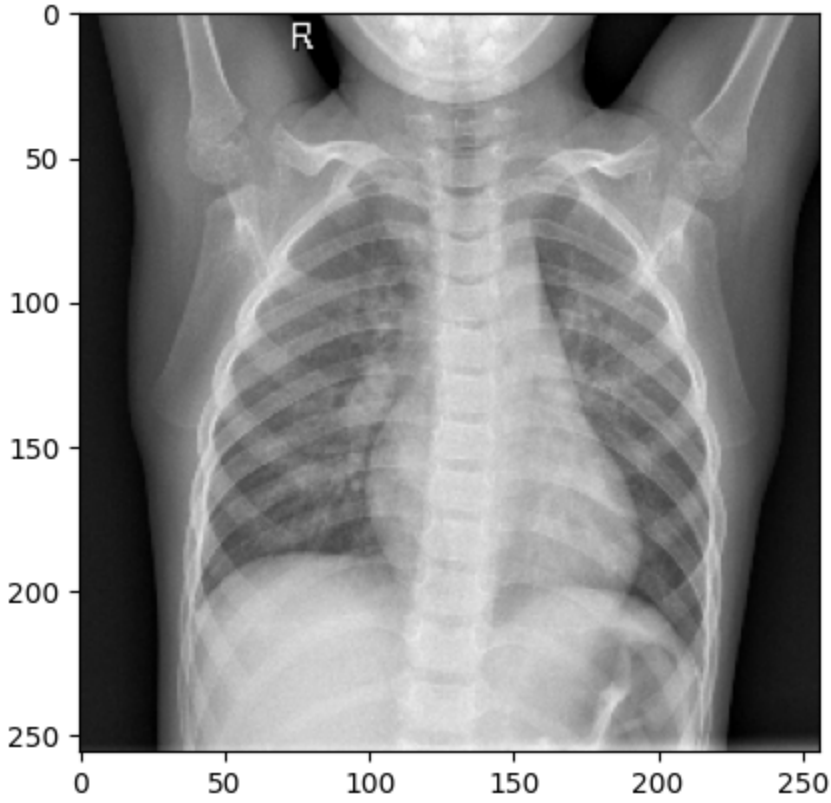
```
(1, 256, 256, 3)
```



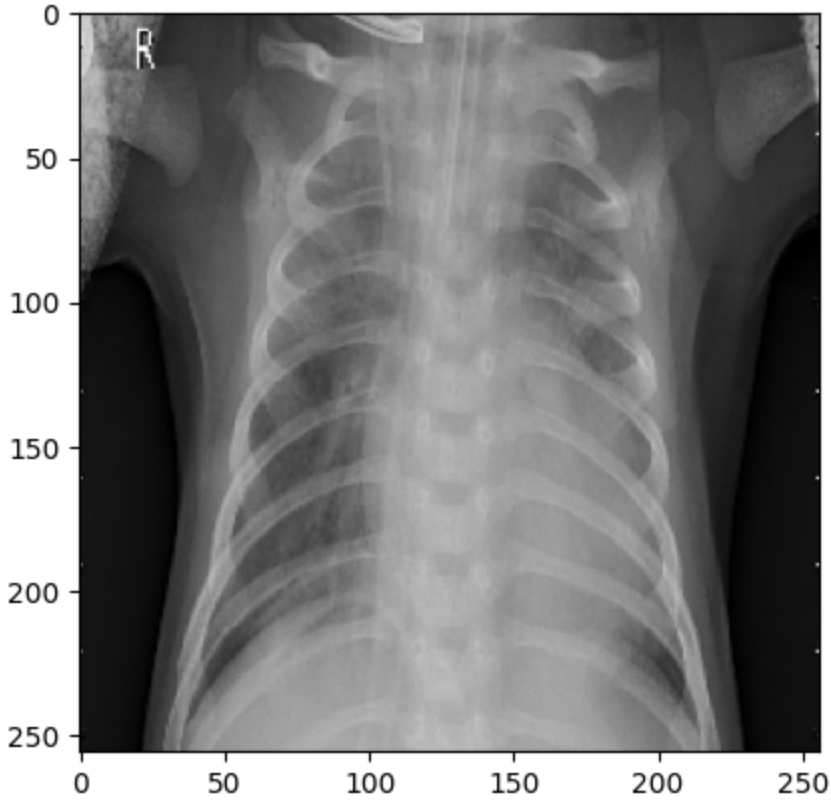
In [ ]:

```
bacteria_img_path = 'chest_xray/test/PNEUMONIA/person82_bacteria_402
virus_img_path = 'chest_xray/test/PNEUMONIA/person11_virus_38.jpeg'
normal_img_path = 'chest_xray/train/NORMAL/IM-0115-0001.jpeg'

#Create tensor for normal Image
normal_img = image.load_img(normal_img_path, target_size=(256, 256))
normal_img_tensor = image.img_to_array(normal_img)
normal_img_tensor = np.expand_dims(normal_img_tensor, axis=0)

#Create tensor for bacterial pneumonia image
bacteria_img = image.load_img(bacteria_img_path, target_size=(256, 2
bacteria_img_tensor = image.img_to_array(bacteria_img)
bacteria_img_tensor = np.expand_dims(bacteria_img_tensor, axis=0)

#Create tensor for viral pneumonia image
virus_img = image.load_img(virus_img_path, target_size=(256, 256))
virus_img_tensor = image.img_to_array(virus_img)
virus_img_tensor = np.expand_dims(virus_img_tensor, axis=0)


#Follow the Original Model Preprocessing
normal_img_tensor /= 255.
bacteria_img_tensor /=255.
virus_img_tensor /=255.

#Check tensor shape
print(normal_img_tensor.shape)
print(bacteria_img_tensor.shape)
print(virus_img_tensor.shape)

#Preview an image
print("Normal Lung")
plt.imshow(normal_img_tensor[0])
plt.show()

print("Bacterial Pneumonia Lung")
plt.imshow(bacteria_img_tensor[0])
plt.show()

print("Viral Pneumonia Lung")
plt.imshow(virus_img_tensor[0])
plt.show()
```

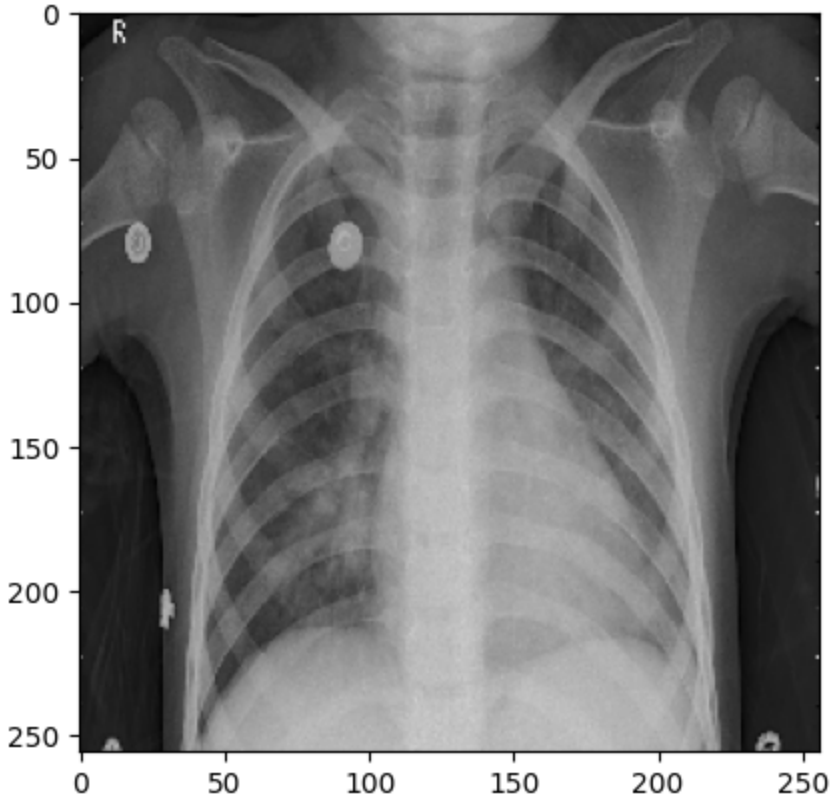
```
(1, 256, 256, 3)
(1, 256, 256, 3)
(1, 256, 256, 3)
Normal Lung
```



Bacterial Pneumonia Lung



Viral Pneumonia Lung

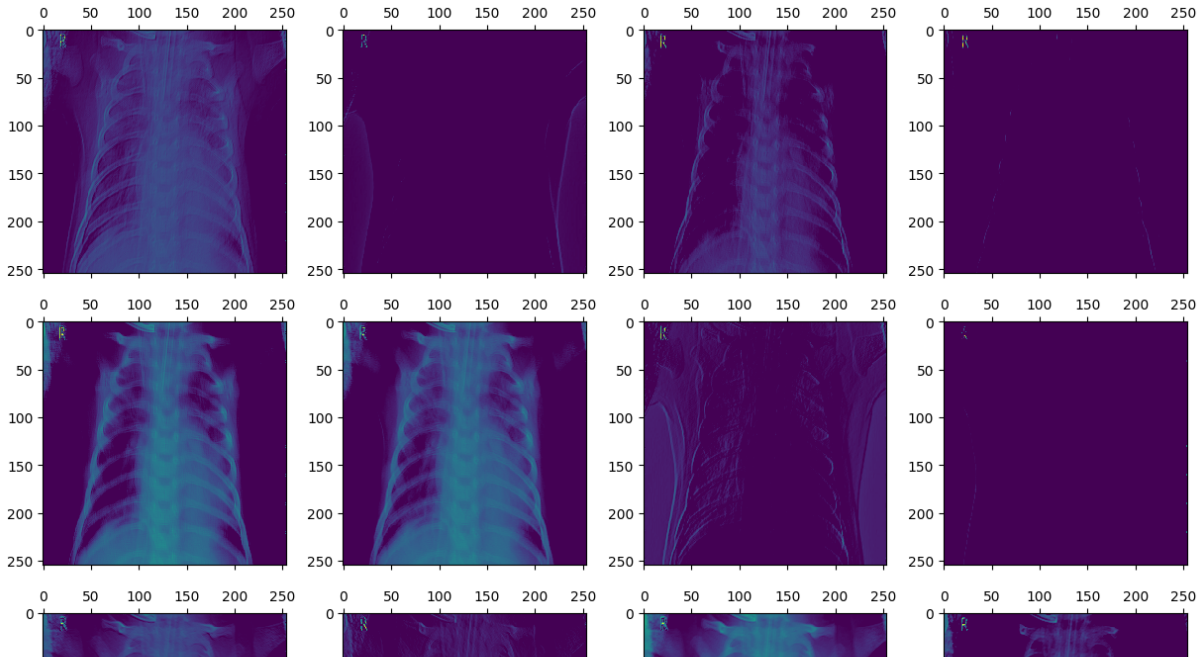


In [ ]:
```
# Extract model layer outputs
layer_outputs = [layer.output for layer in model_cnn_4.layers[:8]]

# Display the models feature maps
activation_model = models.Model(inputs=model_cnn_4.input, outputs=la
```

In [ ]:
```
# Returns an array for each activation layer on the bacteria image
bacterial_activations = activation_model.predict(bacteria_img_tensor
```

```
1/1 [==============================] - 0s 90ms/step
```

In [ ]:
```
fig, axes = plt.subplots(8, 4, figsize=(15,30))
for i in range(32):
    row = i//4
    column = i%4
    ax = axes[row, column]
    bacterial_first_layer_activation = bacterial_activations[0]
    ax.matshow(bacterial_first_layer_activation[0, :, :, i], cmap='v
```

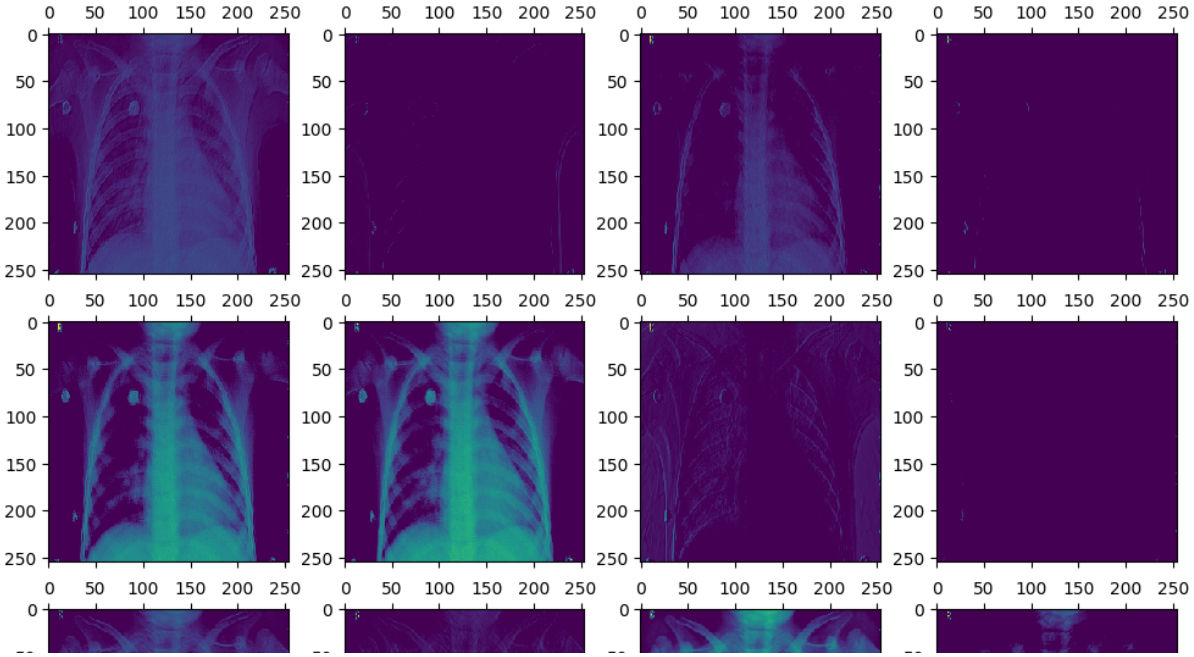


The above images display the feature maps generated by each convolutional layer. These maps identify unique patterns.

In [ ]:
```
# Returns an array for each activation layer on the virus image
viral_activations = activation_model.predict(virus_img_tensor)
```

```
1/1 [==============================] - 0s 31ms/step
```

In [ ]:
```
fig, axes = plt.subplots(8, 4, figsize=(12,24))
for i in range(32):
    row = i//4
    column = i%4
    ax = axes[row, column]
    viral_first_layer_activation = viral_activations[0]
    ax.matshow(viral_first_layer_activation[0, :, :, i], cmap='virid
```



Conclusions

- Further investigate using the model too evaluate pneumonia in chest - xrays. 95% accuracy is very high and this metric means that further investigation is warranted.
- Regularization helped in reducing the variability off the accuracy in later epochs, but reduced the accuracy. These models need to be tested on larger data sets to further investigate the effects off regularization.
- Start running models on a small scale at hospitals.

Future Projects

- Train the model on x-rays in other age groups. Currently, the data is restricted to the under 5 age group. Increase to adults.
- Train the model on other pathogens that can cause pneumonia such as fungi.
- Enhance the model to predict the source off pneumonia (i.e. bacteria, virus, or fungi). This could allow clinicians to administer the appropriate medicines faster.