COVID Project / Yeoman Yoon

Import Libraries and Load Data

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

import cv2 # IMAGE PROCESSING - OPENCV
from glob import glob # FILE OPERATIONS
import itertools
import seaborn as sns

# KERAS AND SKLEARN MODULES
from keras.utils import np_utils
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, GlobalMax
Pooling2D
from keras.layers import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D

from keras.callbacks import ModelCheckpoint,ReduceLROnPlateau,CSVLogger
from keras.optimizers import RMSprop
from keras.optimizers import Adam



from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

from sklearn.preprocessing import LabelBinarizer
```

```
Using TensorFlow backend.
```

In [2]:
```
trainData = pd.read_csv("trainLabels.csv") # use copy to store the original da
ta
testData = pd.read_csv("testLabels.csv")
dataN = trainData.copy()
dataT = testData.copy()
data = pd.concat([dataN, dataT],ignore_index=True) # Will divide the train/tes
t later
```

In [3]:
```
#dataN.sample(10, random_state=2)
#dataT.sample(10, random_state=2)
print(data.shape)
data.sample(10, random_state=2)
```

```
(317, 1)
```

Out[3]:

	Label
264	Viral Pneumonia
190	Normal
7	Viral Pneumonia
192	Normal
189	Normal
261	Viral Pneumonia
215	Normal
126	Covid
106	Covid
275	Covid

In [4]:
```
trainImages = np.load('trainimage.npy')
trainImages.shape
```

Out[4]:
```
(251, 128, 128, 3)
```

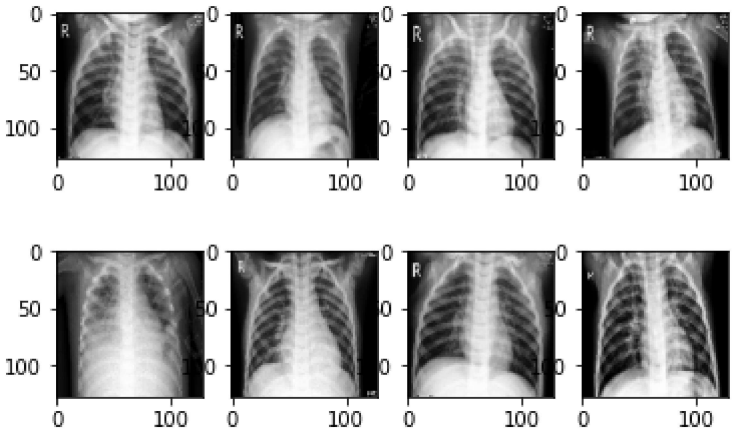
In [5]:
```
testImages = np.load('testimage.npy')
testImages.shape
```

Out[5]:
```
(66, 128, 128, 3)
```

In [6]:
```
images = np.concatenate((trainImages, testImages), axis=0)
images.shape
```

Out[6]:
```
(317, 128, 128, 3)
```

In [7]:
```
for i in range(8):
    plt.subplot(2, 4, i + 1)
    plt.imshow(trainImages[i])
```



EDA

In [8]:
```
data.isnull().sum()
```

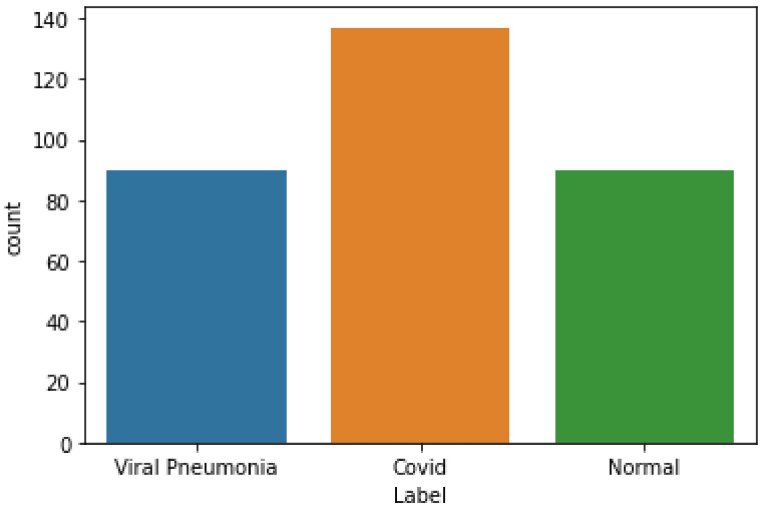
Out[8]:
```
Label    0
dtype: int64
```

In [9]:
```
data["Label"].value_counts()
```

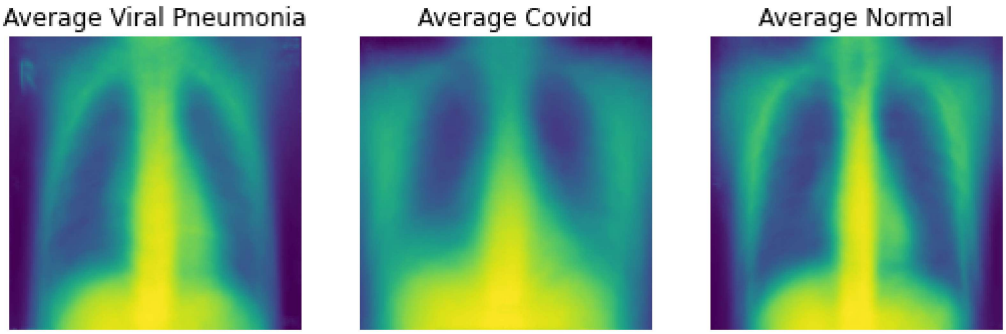
Out[9]:
```
Covid              137
Normal              90
Viral Pneumonia     90
Name: Label, dtype: int64
```

In [10]:
```
sns.countplot(data['Label'])
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c6a6d8e848>
```

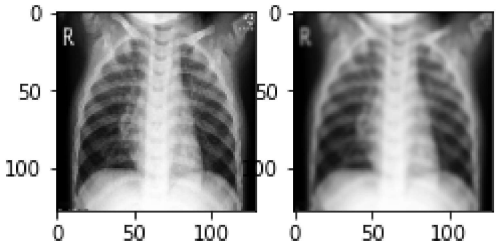


In [11]:
```
def find_mean_img(full_mat):
    # calculate the average
    mean_img = np.mean(full_mat, axis = 0)
    # reshape it back to a matrix
    mean_img = mean_img.reshape((150,150))

    return mean_img

CATEGORIES=data['Label'].unique()
d={ i:[] for i in CATEGORIES}

for i in data.index:
  gray = cv2.cvtColor(images[i], cv2.COLOR_BGR2GRAY)
  gray = cv2.resize(gray,(150,150))
  d[data['Label'][i]].append(gray)

l=[]
for i in d.keys():
  l.append(find_mean_img(d[i]))

plt.subplots(figsize=(12,12))
for i in range(len(l)):
    plt.subplot(3,4,i + 1,title='Average '+list(d.keys())[i])
    plt.imshow(l[i])
    plt.axis('off')
```

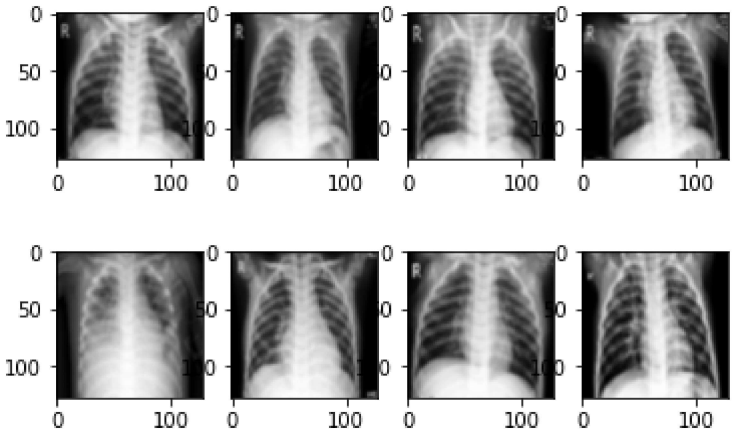


Data Pre-processing

In [12]:
```
newImages = []
sets = []; getEx = True
for i in images:
    blurr = cv2.GaussianBlur(i,(5,5),0) # suggested 5x5
    new = np.zeros_like(i,np.uint8)
    newImages.append(blurr)
    if getEx:
        plt.subplot(2,3,1);plt.imshow(i) # ORIGINAL
        plt.subplot(2,3,2);plt.imshow(blurr) # BLURRED
        plt.show()
        getEx = False
newImages = np.asarray(newImages)
print("# CLEANED IMAGES")

for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(newImages[i])
```



```
# CLEANED IMAGES
```



In [13]:
```
# newImages[0]
#images[0]
#?cv2.GaussianBlur
```

In [14]:
```
# Normalize image data.
newImages = newImages / 255 # run once only
```

In [ ]:
```

```

Make Data Compatible

In [15]:
```
enc = LabelBinarizer()
y = enc.fit_transform(data)
y
```

Out[15]:
```
array([[0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
```

```
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
```

```
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
```

```
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
```

```
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 1, 0],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[0, 0, 1],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
```

```
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0]])
```

In [16]:
```
print(y.shape)
print(newImages.shape)
```

```
(317, 3)
(317, 128, 128, 3)
```

Building CNN

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(newImages,y , test_size=0.
1, random_state=7,stratify=y)
```

In [18]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(285, 128, 128, 3)
(285, 3)
(32, 128, 128, 3)
(32, 3)
```

In [19]:
```
pd.DataFrame(y_train.argmax(axis=1),columns=['label'])['label'].value_counts
()/pd.DataFrame(y_train.argmax(axis=1),columns=['label'])['label'].value_count
s().sum()
```

Out[19]:
```
0    0.431579
2    0.284211
1    0.284211
Name: label, dtype: float64
```

In [20]:
```
pd.DataFrame(y_test.argmax(axis=1),columns=['label'])['label'].value_counts()/
pd.DataFrame(y_test.argmax(axis=1),columns=['label'])['label'].value_counts().
sum()
```

Out[20]:
```
0    0.43750
2    0.28125
1    0.28125
Name: label, dtype: float64
```

In [21]:
```
#?pd.DataFrame
```

In [22]:
```
# Set the CNN model
batch_size=None


model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', batch_input_shape = (batch_size,128, 128,
3)))


model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.4))



model.add(GlobalMaxPooling2D())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(3, activation = "softmax"))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 32)      2432
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 32)      25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
=================================================================
Total params: 338,723
Trainable params: 338,723
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
#Defining thr optimizer and loss function
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
model.compile(optimizer = optimizer, loss = "categorical_crossentropy", metric
s = ["accuracy"])
```

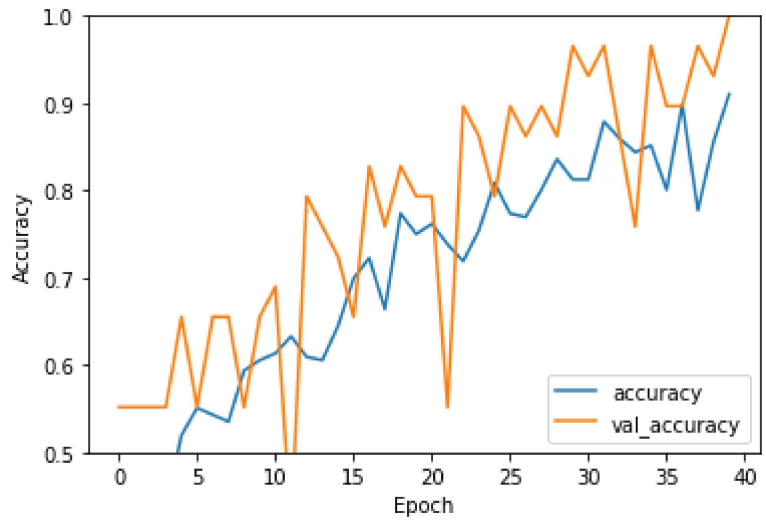
In [24]:
```
# fitting the model with epochs = 40
history=model.fit(X_train, y_train, epochs = 40, validation_split=0.1,batch_si
ze = batch_size)
```

```
Train on 256 samples, validate on 29 samples
Epoch 1/40
256/256 [==============================] - 12s 49ms/step - loss: 1.1448 - acc
uracy: 0.3945 - val_loss: 1.0890 - val_accuracy: 0.5517
Epoch 2/40
256/256 [==============================] - 12s 47ms/step - loss: 1.0937 - acc
uracy: 0.4219 - val_loss: 1.0805 - val_accuracy: 0.5517
Epoch 3/40
256/256 [==============================] - 12s 47ms/step - loss: 1.0923 - acc
uracy: 0.4180 - val_loss: 1.0776 - val_accuracy: 0.5517
Epoch 4/40
256/256 [==============================] - 12s 47ms/step - loss: 1.0491 - acc
uracy: 0.4375 - val_loss: 0.9695 - val_accuracy: 0.5517
Epoch 5/40
256/256 [==============================] - 12s 48ms/step - loss: 1.0673 - acc
uracy: 0.5195 - val_loss: 0.9914 - val_accuracy: 0.6552
Epoch 6/40
256/256 [==============================] - 12s 49ms/step - loss: 0.9511 - acc
uracy: 0.5508 - val_loss: 0.8940 - val_accuracy: 0.5517
Epoch 7/40
256/256 [==============================] - 13s 50ms/step - loss: 0.9762 - acc
uracy: 0.5430 - val_loss: 0.8704 - val_accuracy: 0.6552
Epoch 8/40
256/256 [==============================] - 13s 51ms/step - loss: 0.9413 - acc
uracy: 0.5352 - val_loss: 0.7994 - val_accuracy: 0.6552
Epoch 9/40
256/256 [==============================] - 13s 51ms/step - loss: 0.8831 - acc
uracy: 0.5938 - val_loss: 1.0352 - val_accuracy: 0.5517
Epoch 10/40
256/256 [==============================] - 13s 51ms/step - loss: 0.9268 - acc
uracy: 0.6055 - val_loss: 0.7887 - val_accuracy: 0.6552
Epoch 11/40
256/256 [==============================] - 13s 52ms/step - loss: 0.8834 - acc
uracy: 0.6133 - val_loss: 0.7683 - val_accuracy: 0.6897
Epoch 12/40
256/256 [==============================] - 13s 52ms/step - loss: 0.8764 - acc
uracy: 0.6328 - val_loss: 1.2430 - val_accuracy: 0.4138
Epoch 13/40
256/256 [==============================] - 14s 53ms/step - loss: 0.9444 - acc
uracy: 0.6094 - val_loss: 0.8986 - val_accuracy: 0.7931
Epoch 14/40
256/256 [==============================] - 14s 54ms/step - loss: 0.8719 - acc
uracy: 0.6055 - val_loss: 0.8478 - val_accuracy: 0.7586
Epoch 15/40
256/256 [==============================] - 14s 54ms/step - loss: 0.8247 - acc
uracy: 0.6445 - val_loss: 0.7351 - val_accuracy: 0.7241
Epoch 16/40
256/256 [==============================] - 14s 55ms/step - loss: 0.7487 - acc
uracy: 0.6992 - val_loss: 0.6903 - val_accuracy: 0.6552
Epoch 17/40
256/256 [==============================] - 15s 57ms/step - loss: 0.7290 - acc
uracy: 0.7227 - val_loss: 0.5652 - val_accuracy: 0.8276
Epoch 18/40
256/256 [==============================] - 16s 61ms/step - loss: 0.7936 - acc
uracy: 0.6641 - val_loss: 0.6787 - val_accuracy: 0.7586
Epoch 19/40
256/256 [==============================] - 15s 57ms/step - loss: 0.6098 - acc
```

```
uracy: 0.7734 - val_loss: 0.5388 - val_accuracy: 0.8276
Epoch 20/40
256/256 [==============================] - 15s 58ms/step - loss: 0.5904 - acc
uracy: 0.7500 - val_loss: 0.4746 - val_accuracy: 0.7931
Epoch 21/40
256/256 [==============================] - 17s 66ms/step - loss: 0.6043 - acc
uracy: 0.7617 - val_loss: 0.7294 - val_accuracy: 0.7931
Epoch 22/40
256/256 [==============================] - 16s 61ms/step - loss: 0.6454 - acc
uracy: 0.7383 - val_loss: 0.7503 - val_accuracy: 0.5517
Epoch 23/40
256/256 [==============================] - 15s 57ms/step - loss: 0.6648 - acc
uracy: 0.7188 - val_loss: 0.3651 - val_accuracy: 0.8966
Epoch 24/40
256/256 [==============================] - 15s 57ms/step - loss: 0.5705 - acc
uracy: 0.7539 - val_loss: 0.4643 - val_accuracy: 0.8621
Epoch 25/40
256/256 [==============================] - 15s 57ms/step - loss: 0.5107 - acc
uracy: 0.8086 - val_loss: 0.4394 - val_accuracy: 0.7931
Epoch 26/40
256/256 [==============================] - 15s 59ms/step - loss: 0.5103 - acc
uracy: 0.7734 - val_loss: 0.3573 - val_accuracy: 0.8966
Epoch 27/40
256/256 [==============================] - 15s 57ms/step - loss: 0.5599 - acc
uracy: 0.7695 - val_loss: 0.4470 - val_accuracy: 0.8621
Epoch 28/40
256/256 [==============================] - 15s 57ms/step - loss: 0.5569 - acc
uracy: 0.8008 - val_loss: 0.3917 - val_accuracy: 0.8966
Epoch 29/40
256/256 [==============================] - 15s 57ms/step - loss: 0.4409 - acc
uracy: 0.8359 - val_loss: 0.3274 - val_accuracy: 0.8621
Epoch 30/40
256/256 [==============================] - 16s 61ms/step - loss: 0.4588 - acc
uracy: 0.8125 - val_loss: 0.3559 - val_accuracy: 0.9655
Epoch 31/40
256/256 [==============================] - 15s 60ms/step - loss: 0.5156 - acc
uracy: 0.8125 - val_loss: 0.3492 - val_accuracy: 0.9310
Epoch 32/40
256/256 [==============================] - 15s 58ms/step - loss: 0.3560 - acc
uracy: 0.8789 - val_loss: 0.2835 - val_accuracy: 0.9655
Epoch 33/40
256/256 [==============================] - 15s 57ms/step - loss: 0.3459 - acc
uracy: 0.8594 - val_loss: 0.4342 - val_accuracy: 0.8621
Epoch 34/40
256/256 [==============================] - 15s 57ms/step - loss: 0.3935 - acc
uracy: 0.8438 - val_loss: 0.5185 - val_accuracy: 0.7586
Epoch 35/40
256/256 [==============================] - 15s 57ms/step - loss: 0.3979 - acc
uracy: 0.8516 - val_loss: 0.2225 - val_accuracy: 0.9655
Epoch 36/40
256/256 [==============================] - 15s 57ms/step - loss: 0.4652 - acc
uracy: 0.8008 - val_loss: 0.2753 - val_accuracy: 0.8966
Epoch 37/40
256/256 [==============================] - 15s 57ms/step - loss: 0.2903 - acc
uracy: 0.8984 - val_loss: 0.3234 - val_accuracy: 0.8966
Epoch 38/40
256/256 [==============================] - 15s 57ms/step - loss: 0.4564 - acc
```

```
uracy: 0.7773 - val_loss: 0.3063 - val_accuracy: 0.9655
Epoch 39/40
256/256 [==============================] - 15s 58ms/step - loss: 0.3623 - acc
uracy: 0.8555 - val_loss: 0.2382 - val_accuracy: 0.9310
Epoch 40/40
256/256 [==============================] - 15s 57ms/step - loss: 0.2963 - acc
uracy: 0.9102 - val_loss: 0.1475 - val_accuracy: 1.0000
```

In [25]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right');
```

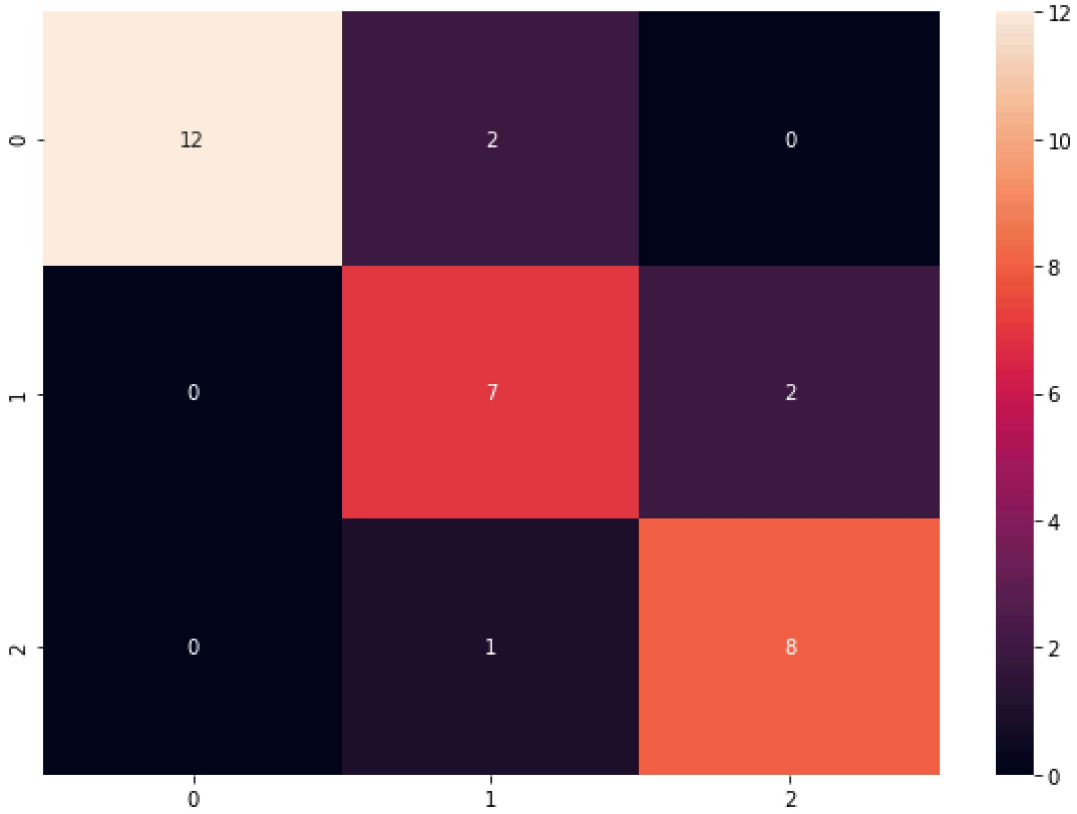


In [26]:
```
# Evaluate the model.

score = model.evaluate(X_test, y_test, verbose=0, batch_size = 19)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.41632675658911467
Test accuracy: 0.84375
```

Confusion Matrix

In [40]:
```
# Predict the values from the validation dataset
Y_pred = model.predict(X_test)
# Convert predictions classes to one hot vectors
result = np.argmax(Y_pred, axis=1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis=1)

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

conf_mat = confusion_matrix(Y_true, result)

df_cm = pd.DataFrame(conf_mat, index = [i for i in range(0, 3)],
                     columns = [i for i in range(0, 3)])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g');
```



In [27]:
```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

epochs = 25
batch_size = 19
```

Data Augmentation (Regularization)

In [28]:
```
# With data augmentation to prevent overfitting (accuracy 0.99286)

datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the da
taset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0
to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction
of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction
of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)
```

In [29]:
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train , test_size=
0.1, random_state=7,stratify=y_train)
```

In [30]:
```
X_train.shape
```

Out[30]:
```
(256, 128, 128, 3)
```

In [31]:
```
# Set the CNN model
batch_size=None
model1 = Sequential()

model1.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', batch_input_shape = (batch_size,128, 128,
3)))


model1.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(Dropout(0.2))


model1.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model1.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'same',
                 activation ='relu'))
model1.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model1.add(Dropout(0.3))

model1.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model1.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model1.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model1.add(Dropout(0.4))



model1.add(GlobalMaxPooling2D())
model1.add(Dense(256, activation = "relu"))
model1.add(Dropout(0.5))
model1.add(Dense(3, activation = "softmax"))
model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 128, 128, 32)      2432
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 128, 128, 32)      25632
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 64, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_7 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
global_max_pooling2d_2 (Glob (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 771
=================================================================
Total params: 338,723
Trainable params: 338,723
Non-trainable params: 0
_________________________________________________________________
```

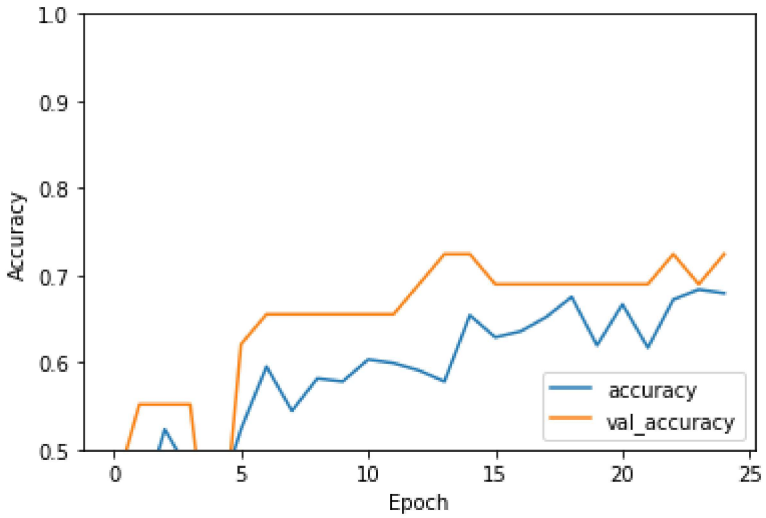
In [32]:
```
#Defining thr optimizer and loss function
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
model1.compile(optimizer = optimizer, loss = "categorical_crossentropy", metri
cs = ["accuracy"])
```

In [33]:
```
#Fitting the model using fit_generator function
#X_train, X_val, y_train, y_test
batch_size = 19
history1 = model1.fit_generator(datagen.flow(X_train,y_train, batch_size=batch
_size),
                                epochs = epochs, validation_data = (X_val,y_va
l),
                                verbose = 2, steps_per_epoch=X_train.shape[0] //
batch_size
                                , callbacks=[learning_rate_reduction])
```

```
Epoch 1/25
 - 14s - loss: 1.1295 - accuracy: 0.3544 - val_loss: 1.0520 - val_accuracy:
0.4483
Epoch 2/25
 - 14s - loss: 1.0446 - accuracy: 0.4219 - val_loss: 1.0611 - val_accuracy:
0.5517
Epoch 3/25
 - 14s - loss: 1.0305 - accuracy: 0.5232 - val_loss: 1.0227 - val_accuracy:
0.5517
Epoch 4/25
 - 14s - loss: 1.0275 - accuracy: 0.4726 - val_loss: 1.0057 - val_accuracy:
0.5517
Epoch 5/25
 - 14s - loss: 1.0808 - accuracy: 0.4304 - val_loss: 1.0937 - val_accuracy:
0.3103

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.000500000023748725
7.
Epoch 6/25
 - 14s - loss: 1.0370 - accuracy: 0.5232 - val_loss: 1.0238 - val_accuracy:
0.6207
Epoch 7/25
 - 14s - loss: 0.9707 - accuracy: 0.5951 - val_loss: 0.9758 - val_accuracy:
0.6552
Epoch 8/25
 - 14s - loss: 1.0040 - accuracy: 0.5443 - val_loss: 0.9685 - val_accuracy:
0.6552
Epoch 9/25
 - 13s - loss: 0.9130 - accuracy: 0.5815 - val_loss: 0.9585 - val_accuracy:
0.6552
Epoch 10/25
 - 14s - loss: 0.9166 - accuracy: 0.5781 - val_loss: 0.8979 - val_accuracy:
0.6552

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.000250000011874362
8.
Epoch 11/25
 - 15s - loss: 0.9070 - accuracy: 0.6032 - val_loss: 0.9194 - val_accuracy:
0.6552
Epoch 12/25
 - 14s - loss: 0.9017 - accuracy: 0.5992 - val_loss: 0.9135 - val_accuracy:
0.6552
Epoch 13/25
 - 15s - loss: 0.8477 - accuracy: 0.5907 - val_loss: 0.8455 - val_accuracy:
0.6897
Epoch 14/25
 - 14s - loss: 0.8601 - accuracy: 0.5781 - val_loss: 0.8545 - val_accuracy:
0.7241
Epoch 15/25
 - 14s - loss: 0.8439 - accuracy: 0.6540 - val_loss: 0.8313 - val_accuracy:
0.7241
Epoch 16/25
 - 14s - loss: 0.8314 - accuracy: 0.6287 - val_loss: 0.8009 - val_accuracy:
0.6897
Epoch 17/25
 - 15s - loss: 0.8166 - accuracy: 0.6356 - val_loss: 0.7823 - val_accuracy:
0.6897
```

```
Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.000125000005937181
4.
Epoch 18/25
 - 13s - loss: 0.8014 - accuracy: 0.6520 - val_loss: 0.8063 - val_accuracy:
0.6897
Epoch 19/25
 - 14s - loss: 0.8096 - accuracy: 0.6751 - val_loss: 0.7703 - val_accuracy:
0.6897
Epoch 20/25
 - 15s - loss: 0.8215 - accuracy: 0.6194 - val_loss: 0.7462 - val_accuracy:
0.6897

Epoch 00020: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-0
5.
Epoch 21/25
 - 14s - loss: 0.7684 - accuracy: 0.6667 - val_loss: 0.7553 - val_accuracy:
0.6897
Epoch 22/25
 - 13s - loss: 0.8454 - accuracy: 0.6167 - val_loss: 0.7687 - val_accuracy:
0.6897
Epoch 23/25
 - 14s - loss: 0.7151 - accuracy: 0.6721 - val_loss: 0.7629 - val_accuracy:
0.7241

Epoch 00023: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-0
5.
Epoch 24/25
 - 14s - loss: 0.7104 - accuracy: 0.6835 - val_loss: 0.7488 - val_accuracy:
0.6897
Epoch 25/25
 - 14s - loss: 0.7293 - accuracy: 0.6793 - val_loss: 0.7689 - val_accuracy:
0.7241
```

In [34]:
```
plt.plot(history1.history['accuracy'], label='accuracy')
plt.plot(history1.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right');
```

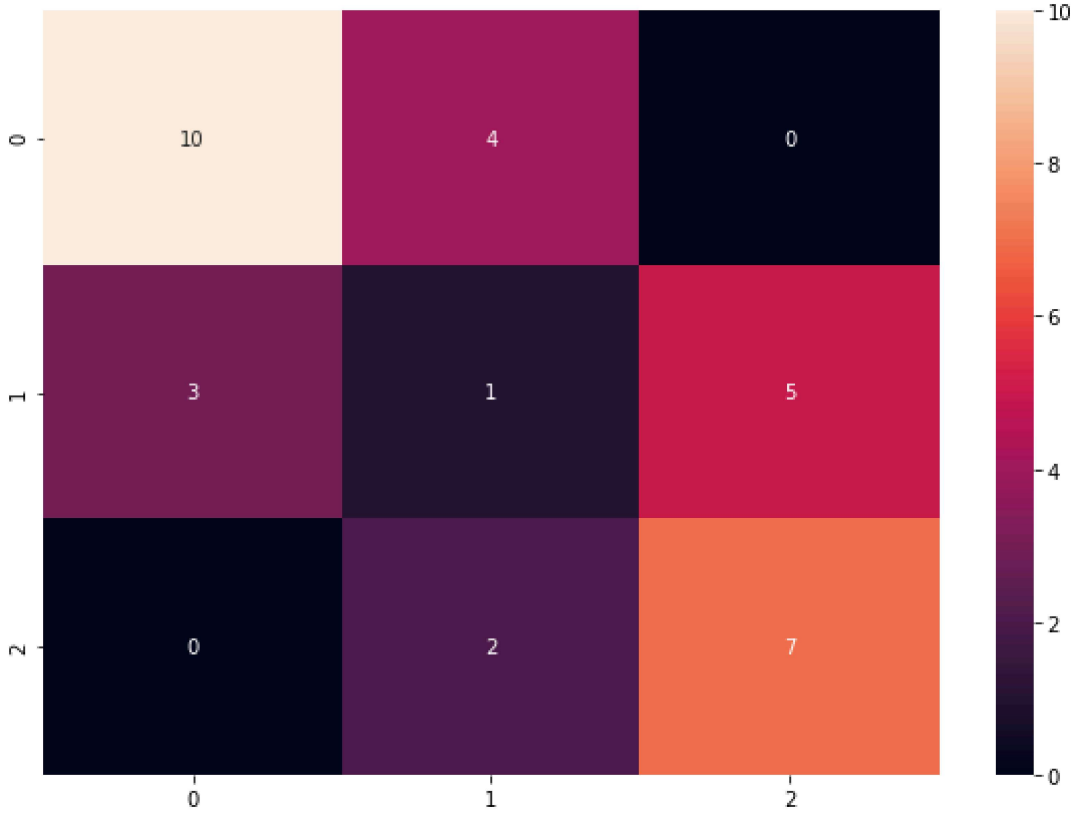


In [35]:
```
# Evaluate the model.

score = model1.evaluate(X_test, y_test, verbose=0, batch_size = 19)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.8379208017140627
Test accuracy: 0.5625
```

Confusion Matrix

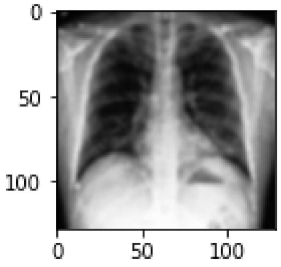
In [41]:
```
# Predict the values from the validation dataset
Y_pred = model1.predict(X_test)
# Convert predictions classes to one hot vectors
result = np.argmax(Y_pred, axis=1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis=1)

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

conf_mat = confusion_matrix(Y_true, result)

df_cm = pd.DataFrame(conf_mat, index = [i for i in range(0, 3)],
                     columns = [i for i in range(0, 3)])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g');
```



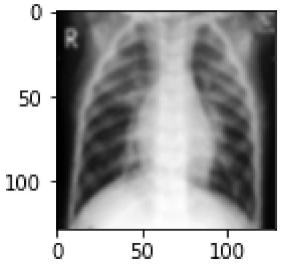
Visualize

Visualize with better model

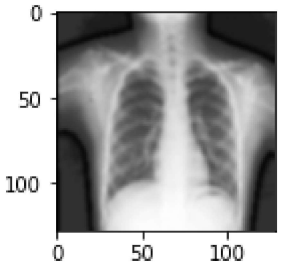
In [39]:
```
plt.figure(figsize=(2,2))
plt.imshow(X_test[2],cmap="gray")
plt.show()
print('Predicted Label', np.argmax(model.predict(X_test[2].reshape(1,128,128,
3))))
print('True Label', np.argmax(y_test[2]))

plt.figure(figsize=(2,2))
plt.imshow(X_test[3],cmap="gray")
plt.show()
print('Predicted Label', np.argmax(model.predict(X_test[3].reshape(1,128,128,
3))))
print('True Label', np.argmax(y_test[3]))

plt.figure(figsize=(2,2)) # 33 doesn't exist pick other randoms.
plt.imshow(X_test[10],cmap="gray")
plt.show()
print('Predicted Label', np.argmax(model.predict(X_test[10].reshape(1,128,128,
3))))
print('True Label', np.argmax(y_test[10]))

plt.figure(figsize=(2,2))
plt.imshow(X_test[16],cmap="gray")
plt.show()
print('Predicted Label', np.argmax(model.predict(X_test[16].reshape(1,128,128,
3))))
print('True Label', np.argmax(y_test[16]))

plt.figure(figsize=(2,2))
plt.imshow(X_test[31],cmap="gray")
plt.show()
print('Predicted Label', np.argmax(model.predict(X_test[31].reshape(1,128,128,
3))))
print('True Label', np.argmax(y_test[31]))
```



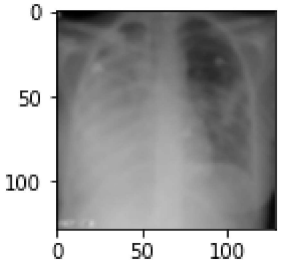
```
Predicted Label 1
True Label 0
```



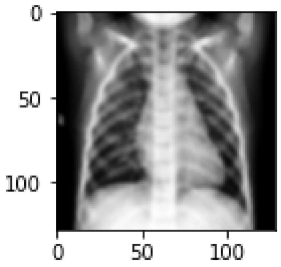
```
Predicted Label 2
True Label 2
```



```
Predicted Label 1
True Label 1
```



```
Predicted Label 0
True Label 0
```



```
Predicted Label 2
True Label 1
```

Conclusion:

epoch 40 is good for avoiding overfitting. Tried Regularization but didn't improve the model.