In [1]:
```
#Importing packages

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib import image
from keras.models import Sequential
from keras import layers
from keras.layers import Flatten, Dense, Dropout, Activation
# from keras.layers.normalization import BatchNormalization
# from keras.layers.convolutional import Conv2D
from keras.models import Model
from keras.optimizers import SGD, Adam
from keras.losses import categorical_crossentropy
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import os
from shutil import move


import csv
```

```
Using TensorFlow backend.
```

Type *Markdown* and LaTeX: α^2

Splitting the pneumonia into viral and bacteria foldersfolders

In [2]:
```
#Splitting the pneumonia into viral and bacteria foldersfolders
train_location="chest_xray/train"
test_location="chest_xray/test"
orig_train_pneumonia="chest_xray/train/PNEUMONIA"
orig_test_pneumonia="chest_xray/test/PNEUMONIA"
```

In [3]:
```
#Definition to move viral and bacteria pneumonia images into differenct directory
#Purpose is to be able to use ImageDataGenerator.flow_from_direcotry()
def reorganize_files(pneumonia_directory, parent_directory):
    bac_dir=parent_directory + "/Bacterial"
    vir_dir=parent_directory + "/Viral"

    os.mkdir(bac_dir)
    os.mkdir(vir_dir)

    for filename in os.listdir(pneumonia_directory):
        if (filename.lower().find("bacteria") == -1): #Did not contain bacteria i
            move(pneumonia_directory+"/"+filename,vir_dir)
        else: #This is bacterial pneumonia.
            move(pneumonia_directory+"/"+filename,bac_dir)

    os.rmdir(pneumonia_directory)
```

In [4]:
```
#Check if the folder were indeed been organized., if not, move it.

if(os.path.exists(orig_train_pneumonia)):
    reorganize_files(orig_train_pneumonia, train_location)

if(os.path.exists(orig_test_pneumonia)):
    reorganize_files(orig_test_pneumonia, test_location)
```

Preview the Images

In [5]:
```
#Load a test image and see everything loaded

location = "chest_xray/train/NORMAL"
location_b = "chest_xray/train/Bacterial"
location_v ="chest_xray/train/Viral"

train_normal_example = plt.imread(location + "/IM-0311-0001.jpeg")
train_bacterial_example = plt.imread(location_b + "/person1000_bacteria_2931.jpeg
train_Viral_example = plt.imread(location_v + "/person1000_virus_1681.jpeg")
#As this is black and white image, it only has one dimension.

print(train_normal_example.shape)
print(train_bacterial_example.shape)
print(train_Viral_example.shape)
```

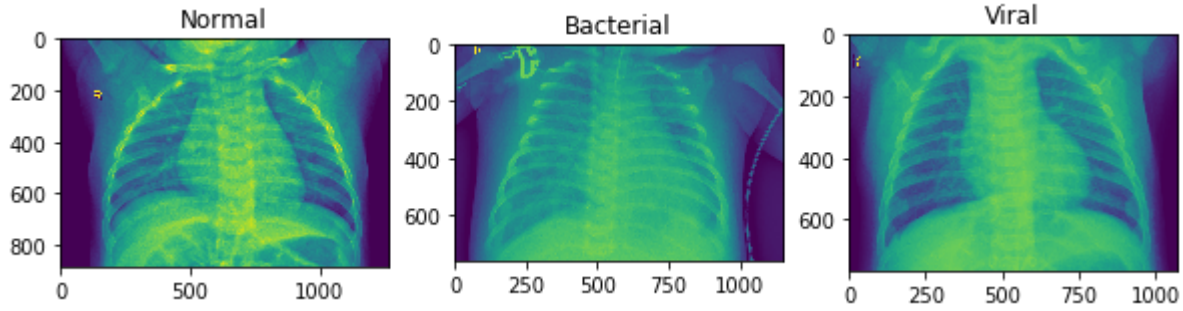
```
(885, 1268)
(760, 1152)
(768, 1072)
```

In [6]:
```
f = plt.figure(figsize= (10,6))
a0 = f.add_subplot(1, 3, 1)
img_plot = plt.imshow(train_normal_example)
a0.set_title('Normal')

a1 = f.add_subplot(1,3,2)
img_plot = plt.imshow(train_bacterial_example)
a1.set_title('Bacterial')

a2 = f.add_subplot(1, 3, 3)
img_plot = plt.imshow(train_Viral_example)
a2.set_title('Viral')
```

Out[6]: Text(0.5, 1.0, 'Viral')



Load dataset

In [7]:
```
from keras.preprocessing.image import ImageDataGenerator
train_datagen=ImageDataGenerator(rescale=1./255,
                                 height_shift_range=0.2,
                                 width_shift_range=0.2,
                                 horizontal_flip = True)

test_datagen=ImageDataGenerator(rescale=1./255)

image_classes=["Bacterial", "NORMAL", "Viral"]


training_set = train_datagen.flow_from_directory(train_location,
                                                 target_size = (224, 224), #For t
                                                 batch_size = 32,
                                                 classes=image_classes,
                                                 shuffle=True,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_location,
                                                 target_size = (224, 224),
                                                 batch_size = 24,
                                                 classes=image_classes,
                                                 shuffle=False,
                                                 class_mode = 'categorical')
```

```
Found 5216 images belonging to 3 classes.
Found 624 images belonging to 3 classes.
```

Build CNN

In [ ]:

In [ ]:

data augmentation

transfer learning

Analysis after model building: confusion matrix, precision, recall, f1 score

In [8]:
```
#Getting VGG19

from keras.applications import VGG19, VGG16, ResNet50

# VGG19_base_model = VGG19(weights="imagenet", input_shape=(224,224,3))
# VGG19_base_model.summary()

#resnet_base_model=ResNet50(weights="imagenet", input_shape=(224,224,3))
# resnet_base_model.summary()
```

In [9]:
```
#Making a class to make it easier to fine tune models.

class modelCreator:
    def __init__(self, model_name,base_model, Dense_Layer_Configurations, AddDrop
        self.model_name=model_name
        self.base_model=base_model
        self.Dense_Layer_Configurations=Dense_Layer_Configurations
        self.AddDropOut=AddDropOut
        self.Dropout_Float=Dropout_Float
        self.Freeze_Model_Layer=Freeze_Model_Layer

    def configureModel(self, lastLayer):
        previousLayer=lastLayer
        index=0

        for neurons in self.Dense_Layer_Configurations[:-1]: #Do not include the
            fc=Dense(neurons, activation='relu')(previousLayer)
            index+=1
            previousLayer=fc
            if(self.AddDropOut):
                fc=Dropout(self.Dropout_Float)(previousLayer)
                previousLayer=fc
                index+=1

        #Adding softmax
        softmaxLayer=Dense(3, activation='softmax')(previousLayer)
        index+=1
        self.model=Model(self.base_model.inputs,softmaxLayer)

        if(self.Freeze_Model_Layer): #This is to determine whether we should only
            for layer in self.model.layers[:-1*index]:  #Only go up to last flatt
                layer.trainable=False

    def combineModels(self):
        #Freeze models
        if(self.model_name.lower() == "vgg19" or self.model_name.lower() == "vgg1

            baseOutput=self.base_model.get_layer("block5_pool").output #Block5_po
            flatten=Flatten()(baseOutput) #Add the flatten layer

            self.configureModel(flatten)

#             for neurons in self.Dense_Layer_Configurations[:-1]:
#                 fc=Dense(neurons, activation='relu')(previousLayer)
#                 index+=1
#                 if(self.AddDropOut):
#                     fc=Dropout(self.Dropout_Float)(fc)
#                     index+=1
#                 previousLayer=fc

#             softmaxLayer=Dense(3, activation='softmax')(previousLayer)
#             index+=1
#             self.model=Model(self.base_model.inputs,softmaxLayer)

#             if(self.Freeze_Model_Layer):
#                 for layer in self.model.layers[:-1*index]:
```

```
#                     layer.trainable=False
            return self.model

        elif(self.model_name.lower()=="resnet"): #too bad python doesn't have swi
            baseOutput=self.base_model.layers[-3].output
            flatten=Flatten()(baseOutput)
            self.configureModel(flatten)
            return self.model
        else:
            print("Please double check your model name and try again")
```

In [10]:
```
File_Name = "CNN_Results.csv"
ResultsDict = {}

Column_Header = ["Model Name", "Layer Configurations", "Dropout Enabled", "Dropou

with open(File_Name, 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(Column_Header)
```

In [11]:
```
#Compile Model Method
class compileModel:
    def __init__(self,model, optimizer_name="sgd", learning_rate=0.01, decay=0.1,
        self.model=model
        self.optimizer_name=optimizer_name
        self.learning_rate=learning_rate
        self.decay=decay
        self.momentum=momentum
        self.metrics=metrics

    def compile_model(self):
        if(self.optimizer_name.lower() == "sgd"):
            self.optimizer=SGD(learning_rate=self.learning_rate, momentum=self.mo
        else:
            self.optimizer="Adam"

        self.model.compile(loss="categorical_crossentropy", optimizer=self.optimi
        return self.model
```

In [12]:
```
def FitModel(model, number_of_batches, validation_batches, epoch, show_progress=F

    if(show_progress):
        show=1
    else:
        show=0

    history = model.fit_generator(generator=training_set,
                                  steps_per_epoch=number_of_batches,
                                  epochs=epoch,
                                  verbose=show,
                                  validation_data=test_set,
                                  validation_steps=validation_batches)

    return history
```

In [13]:
```
# modelObj = modelCreator(model_name="VGG19",
#                             base_model=VGG19_base_model,
#                             Dense_Layer_Configurations=[1000,400,150,70,20,6,3],
#                             AddDropOut=True,
#                             Dropout_Float=0.2,
#                             Freeze_Model_Layer=True)

# model=modelObj.combineModels()

# modelCompiler=compileModel(model=model)
# model=modelCompiler.compile_model()


# history=FitModel(model, 60, 10, 100,True)
```

In [14]:
```
def writeFilesAndSaveDictionary(ModelDict, ScoreDict, DictKey, ScoreList):
    ResultsDict[DictKey] = ScoreList

    with open(File_Name, 'a', newline='') as csvfile:
        file_writer = csv.writer(csvfile)
        neurons = ""
        for neuron in ModelDict["Layer Configurations"]: #can't simply go to toSt
            neurons += str(neuron) + "_"

        ModelDict["Layer Configurations"] = neurons

        row = []

        for i in ModelDict:
            row.append(str(ModelDict[i]))

        for j in ScoreDict:
            row.append(str(ScoreDict[j]))

        file_writer.writerow(row)
```

In [16]:
```
#Try out different parameters.
XferLearningModels = ["VGG19", "VGG16"]
#XferLearningModels = ["ResNet"] #-- Ran out of memory for ResNet
DenseConfigurations = [[500,150,70,20,6,3],
                       [700, 200, 80, 30, 10,3],
                       [1500,500,150,50,20,8,3],
                      [2000,1000,500,250,125,40,10,3]]
Dropouts = [True, False]
DropoutFloat = [0.2,0.5,0.3,0.4]
Freeze_Model_Layer = [True, False]

Optimizers=["Adam", "sgd"]

Batches=[20,30,50,100]
Epochs= [30,50,80]
```

In [ ]:
```

```

In [15]:
```
def DictKeyGenerator(list):
    key = ""
    for i in list:
        key += str(i) + " "

    return key
```

In [18]:
```
#Tuning Evertyhing accorid

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from random import randint

for i in range(0,200): #200 models are way too much.  Ran out of memory.

    #Due to time and budget constraint, cannot search through every model combinc
    XferLearningModels_Index = randint(0,len(XferLearningModels)-1)
    DenseConfigurations_Index = randint(0,len(DenseConfigurations)-1)
    Dropouts_Index = randint(0,len(Dropouts)-1)
    DropoutFloat_Index = randint(0,len(DropoutFloat)-1)
    Freeze_Model_Layer_Index=randint(0,len(Freeze_Model_Layer)-1)
    Optimizers_Index=randint(0,len(Optimizers)-1)
    Batches_Index=randint(0,len(Batches)-1)
    Epochs_Index=randint(0,len(Epochs)-1)

    if(XferLearningModels[XferLearningModels_Index].lower() == "vgg19"):
        base_model = VGG19(weights="imagenet", input_shape=(224,224,3))
    elif(XferLearningModels[XferLearningModels_Index].lower()=="vgg16"):
        base_model = VGG16(weights="imagenet", input_shape=(224,224,3))
    else:
        base_model = ResNet50(weights="imagenet", input_shape=(224,224,3))

    #Using the above
    Model_Settings = {}
    Model_Settings["Model Name"] = XferLearningModels[XferLearningModels_Index]
    Model_Settings["Layer Configurations"] = DenseConfigurations[DenseConfigurati
    Model_Settings["Dropout Enabled"] = Dropouts[Dropouts_Index]
    Model_Settings["Dropout Percentage"] = DropoutFloat[DropoutFloat_Index]
    Model_Settings["Froze PreTrained Model Weights"] = Freeze_Model_Layer[Freeze_
    Model_Settings["Optimizer"] = Optimizers[Optimizers_Index]
    Model_Settings["Batch"] = Batches[Batches_Index]
    Model_Settings["Epoch"] = Epochs[Epochs_Index]

    #Using the above to get the key to input into the ResultsDict {}
    keyCombination = [XferLearningModels[XferLearningModels_Index],
                      DenseConfigurations[DenseConfigurations_Index],
                      Dropouts[Dropouts_Index],
                      DropoutFloat[DropoutFloat_Index],
                      Freeze_Model_Layer[Freeze_Model_Layer_Index],
                      Optimizers[Optimizers_Index],
                      Batches[Batches_Index],
                      Epochs[Epochs_Index]]

    DictKey=DictKeyGenerator(keyCombination)

    #Make sure this model hasn't been run before.
    if(ResultsDict.get(DictKey) != None): #This model has already been run.
        continue


    #Creating Model
```

```
    modelObj = modelCreator(model_name=XferLearningModels[XferLearningModels_Inde
                    base_model=base_model,
                    Dense_Layer_Configurations=DenseConfigurations[DenseConfigura
                    AddDropOut=Dropouts[Dropouts_Index],
                    Dropout_Float=DropoutFloat[DropoutFloat_Index],
                    Freeze_Model_Layer=Freeze_Model_Layer[Freeze_Model_Layer_Inde

    model=modelObj.combineModels()

    #Compile Model
    modelCompiler=compileModel(model=model, optimizer_name=Optimizers[Optimizers_
    model=modelCompiler.compile_model()

    #Fitting Model
    history = FitModel(model, Batches[Batches_Index], 26, Epochs[Epochs_Index], T

    #Make the prediction
    y_predict = model.predict_generator(generator=test_set, steps=26)

    #Un-One hot encode and just use label encode
    predict_labels=y_predict.argmax(axis=1)
    test_labels = test_set.classes

    #Get all the scores.  Macro indicate a non-weighted average.  Hence it would
    ac=accuracy_score(test_labels, predict_labels)
    ps=precision_score(test_labels, predict_labels, average="macro")
    fs=f1_score(test_labels, predict_labels, average="macro")
    rs=recall_score(test_labels, predict_labels, average="macro")

    Scores = {}
    Scores["Acc_Score"] = ac
    Scores["Pre_Score"] = ps
    Scores["F1_Score"] = fs
    Scores["Rec_Score"] = rs
    Scores_list = [ac,ps,fs,rs]

    writeFilesAndSaveDictionary(Model_Settings, Scores, DictKey, Scores_list)
```

```
Epoch 7/30
30/30 [==============================] - 23s 780ms/step - loss: 1.2338 - accu
racy: 0.4479 - val_loss: 1.1739 - val_accuracy: 0.3878
Epoch 8/30
30/30 [==============================] - 23s 781ms/step - loss: 1.3384 - accu
racy: 0.4135 - val_loss: 1.1775 - val_accuracy: 0.3878
Epoch 9/30
30/30 [==============================] - 23s 775ms/step - loss: 1.2345 - accu
racy: 0.4333 - val_loss: 1.1841 - val_accuracy: 0.3878
Epoch 10/30
30/30 [==============================] - 23s 772ms/step - loss: 1.2128 - accu
racy: 0.4479 - val_loss: 1.1955 - val_accuracy: 0.3878
Epoch 11/30
30/30 [==============================] - 24s 796ms/step - loss: 1.1566 - accu
racy: 0.4646 - val_loss: 1.2086 - val_accuracy: 0.3878
Epoch 12/30
30/30 [==============================] - 23s 777ms/step - loss: 1.1792 - accu
racy: 0.4396 - val_loss: 1.2184 - val_accuracy: 0.3878
Epoch 13/30
```

```
30/30 [==============================] - 24s 785ms/step - loss: 1.1356 - accu
```

In [16]:

```
winningModelobj = modelCreator(model_name="VGG16",
                  base_model=VGG16(weights="imagenet", input_shape=(224,224,3))
                  Dense_Layer_Configurations=[700, 200, 80, 30, 10,3],
                  AddDropOut=False,
                  Dropout_Float=0.5,
                  Freeze_Model_Layer=True)

winningModel=winningModelobj.combineModels()

#Compile Model
winningModelCompiler=compileModel(model=winningModel, optimizer_name="sgd")
winningModel=winningModelCompiler.compile_model()

#Fitting Model
history = FitModel(winningModel, 50, 26, 80, True)

#Make the prediction
y_predict = winningModel.predict_generator(generator=test_set, steps=26)

#Un-One hot encode and just use label encode
predict_labels=y_predict.argmax(axis=1)
test_labels = test_set.classes
```

```
racy: 0.7644 - val_loss: 0.4641 - val_accuracy: 0.8574
Epoch 75/80
50/50 [==============================] - 35s 693ms/step - loss: 0.5235 - accu
racy: 0.7775 - val_loss: 1.4744 - val_accuracy: 0.6955
Epoch 76/80
50/50 [==============================] - 35s 695ms/step - loss: 0.5211 - accu
racy: 0.7875 - val_loss: 0.3268 - val_accuracy: 0.7644
Epoch 77/80
50/50 [==============================] - 34s 688ms/step - loss: 0.5039 - accu
racy: 0.7856 - val_loss: 0.6070 - val_accuracy: 0.8349
Epoch 78/80
50/50 [==============================] - 35s 701ms/step - loss: 0.5102 - accu
racy: 0.7781 - val_loss: 0.5573 - val_accuracy: 0.7885
Epoch 79/80
50/50 [==============================] - 36s 714ms/step - loss: 0.4943 - accu
racy: 0.7819 - val_loss: 0.7154 - val_accuracy: 0.8045
Epoch 80/80
50/50 [==============================] - 35s 705ms/step - loss: 0.5071 - accu
racy: 0.7856 - val_loss: 0.5002 - val_accuracy: 0.8526
```

In [17]:
```
winningModel.summary()
```

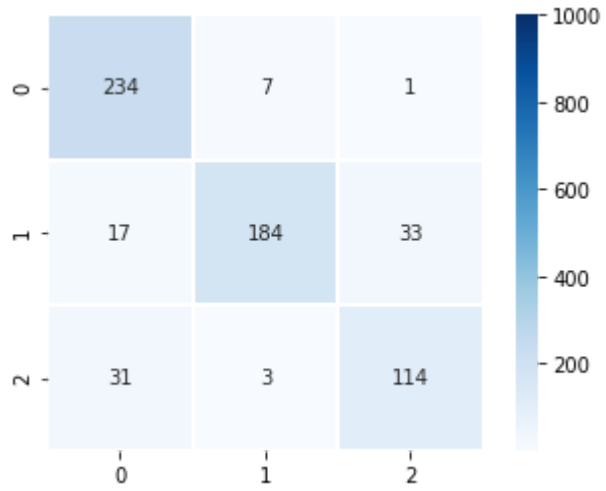
```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 700)               17562300
_________________________________________________________________
dense_2 (Dense)              (None, 200)               140200
_________________________________________________________________
dense_3 (Dense)              (None, 80)                16080
_________________________________________________________________
dense_4 (Dense)              (None, 30)                2430
_________________________________________________________________
dense_5 (Dense)              (None, 10)                310
_________________________________________________________________
```

```
dense_6 (Dense)              (None, 3)                 33
=================================================================
Total params: 32,436,041
Trainable params: 17,721,353
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [24]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix

confusion_matrix=confusion_matrix(test_labels, predict_labels)

matrix=sns.heatmap(confusion_matrix,linewidths=1,vmax=1000,
square=True, cmap="Blues",annot=True, fmt="1")
```



In [28]:
```
from sklearn.metrics import classification_report
target_names = ['Normal', 'Bacterial', 'Viral']

ac=accuracy_score(test_labels, predict_labels)
print("Test set accuracy: " + str(round(ac,4))

print(classification_report(test_labels, predict_labels, target_names=target_name
```

```
Test set accuracy: 0.8525641025641025
              precision    recall  f1-score   support

      Normal       0.83      0.97      0.89       242
   Bacterial       0.95      0.79      0.86       234
       Viral       0.77      0.77      0.77       148

    accuracy                           0.85       624
   macro avg       0.85      0.84      0.84       624
weighted avg       0.86      0.85      0.85       624
```

In [ ]:

In [ ]:

In [ ]: