Class Challenge: Image Classification of COVID-19 X-rays

Task 1 [Total points: 30]

Setup

- This assignment involves the following packages: 'matplotlib', 'numpy', and 'sklearn'.
- If you are using conda, use the following commands to install the above packages:

```
conda install matplotlib
conda install numpy
conda install -c anaconda scikit-learn
```

- If you are using pip, use use the following commands to install the above packages:

```
pip install matplotlib
pip install numpy
pip install sklearn
```

Data

Please download the data using the following link: [COVID-19 (https://drive.google.com/file/d/1Y88tgqpQ1Pjko_7rntcPowOJs_QNOrJ-/view)](https://drive.google.com/file/d/1Y88tgqpQ1Pjko_7rntcPowOJs_QNOrJ-/view).

- After downloading 'Covid_Data_GradientCrescent.zip', unzip the file and you should see the following data structure:

|--all
|--------train
|--------test
|--two
|--------train
|--------test

- Put the 'all' folder, the 'two' folder and this python notebook in the **same directory** so that the following code can correctly locate the data.

[20 points] Binary Classification: COVID-19 vs. Normal

In [1]:
```
import os

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

os.environ['OMP_NUM_THREADS'] = '1'
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
tf.__version__
```

Out[1]: '2.3.0'

Load Image Data

In [2]:
```
DATA_LIST = os.listdir('Covid_Data_GradientCrescent/two/train')
DATASET_PATH  = 'Covid_Data_GradientCrescent/two/train'
TEST_DIR =  'Covid_Data_GradientCrescent/two/test'
IMAGE_SIZE    = (224, 224)
NUM_CLASSES   = len(DATA_LIST)
BATCH_SIZE    = 15  # try reducing batch size or freeze more layers if y
our GPU runs out of memory
NUM_EPOCHS    = 40
LEARNING_RATE = 0.0005 # start off with high rate first 0.001 and experi
ment with reducing it gradually
```

Generate Training and Validation Batches

In [3]:
```
train_datagen = ImageDataGenerator(rescale=1./255,rotation_range=50,feat
urewise_center = True,
                                   featurewise_std_normalization = True,
width_shift_range=0.2,
                                   height_shift_range=0.2,shear_range=0.
25,zoom_range=0.1,
                                   zca_whitening = True,channel_shift_ra
nge = 20,
                                   horizontal_flip = True,vertical_flip
= True,
                                   validation_split = 0.2,fill_mode='con
stant')

train_batches = train_datagen.flow_from_directory(DATASET_PATH,target_si
ze=IMAGE_SIZE,
                                                  shuffle=True,batch_siz
e=BATCH_SIZE,
                                                  subset = "training",se
ed=42,
                                                  class_mode="binary")

valid_batches = train_datagen.flow_from_directory(DATASET_PATH,target_si
ze=IMAGE_SIZE,
                                                  shuffle=True,batch_siz
e=BATCH_SIZE,
                                                  subset = "validation",
seed=42,
                                                  class_mode="binary")
```

```
Found 104 images belonging to 2 classes.
Found 26 images belonging to 2 classes.

/Applications/anaconda3/lib/python3.8/site-packages/keras_preprocessin
g/image/image_data_generator.py:342: UserWarning: This ImageDataGenerat
or specifies `zca_whitening` which overrides setting of`featurewise_std
_normalization`.
  warnings.warn('This ImageDataGenerator specifies '
```

[10 points] Build Model of SGD+momentum+nesterov optimizer Model

Hint: Starting from a pre-trained model typically helps performance on a new task, e.g. starting with weights obtained by training on ImageNet.

In [4]:
```
from tensorflow.keras import models, layers, optimizers

vgg16 = tf.keras.applications.VGG16(weights="imagenet", include_top=Fals
e, input_shape=(224,224,3))

model_sgd = tf.keras.models.Sequential([
    vgg16,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(256, activation="relu", name='dense_feature'),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(1, activation="sigmoid")
])

model_sgd.compile(loss="binary_crossentropy",
              optimizer=optimizers.SGD(lr=LEARNING_RATE, momentum=0.9, n
esterov=True),
              metrics=['accuracy'])

model_sgd.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688  
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0         
_________________________________________________________________
dropout (Dropout)            (None, 25088)             0         
_________________________________________________________________
dense_feature (Dense)        (None, 256)               6422784   
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense (Dense)                (None, 1)                 257       
=================================================================
Total params: 21,137,729
Trainable params: 21,137,729
Non-trainable params: 0
_________________________________________________________________
```

[5 points] Train Model of SGD+momentum+nesterov optimizer Model

In [42]:

```
#FIT MODEL
print(len(train_batches))
print(len(valid_batches))

STEP_SIZE_TRAIN=train_batches.n//train_batches.batch_size
STEP_SIZE_VALID=valid_batches.n//valid_batches.batch_size

import time
start_time = time.time()

output = model_sgd.fit(train_batches,
                 validation_data = valid_batches,
                 validation_steps = STEP_SIZE_VALID,
                 batch_size = BATCH_SIZE,
                 steps_per_epoch =STEP_SIZE_TRAIN,
                 epochs= NUM_EPOCHS)

end_time = time.time()
print(f"The execution time is: {end_time-start_time} seconds")
```

```
7
2

/Applications/anaconda3/lib/python3.8/site-packages/keras_preprocessin
g/image/image_data_generator.py:720: UserWarning: This ImageDataGenerat
or specifies `featurewise_center`, but it hasn't been fit on any traini
ng data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
/Applications/anaconda3/lib/python3.8/site-packages/keras_preprocessin
g/image/image_data_generator.py:739: UserWarning: This ImageDataGenerat
or specifies `zca_whitening`, but it hasn't been fit on any training da
ta. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
Epoch 1/40
6/6 [==============================] - 71s 12s/step - loss: 0.8278 - ac
curacy: 0.5169 - val_loss: 0.6528 - val_accuracy: 0.6000
Epoch 2/40
6/6 [==============================] - 69s 12s/step - loss: 0.7704 - ac
curacy: 0.5169 - val_loss: 0.5584 - val_accuracy: 1.0000
Epoch 3/40
6/6 [==============================] - 75s 13s/step - loss: 0.5990 - ac
curacy: 0.7191 - val_loss: 0.3947 - val_accuracy: 0.8000
Epoch 4/40
6/6 [==============================] - 68s 11s/step - loss: 0.4543 - ac
curacy: 0.7416 - val_loss: 0.2879 - val_accuracy: 0.9333
Epoch 5/40
6/6 [==============================] - 70s 12s/step - loss: 0.3854 - ac
curacy: 0.8202 - val_loss: 0.0392 - val_accuracy: 1.0000
Epoch 6/40
6/6 [==============================] - 70s 12s/step - loss: 0.2468 - ac
curacy: 0.9000 - val_loss: 0.1442 - val_accuracy: 0.9333
Epoch 7/40
6/6 [==============================] - 70s 12s/step - loss: 0.1787 - ac
curacy: 0.9333 - val_loss: 0.1176 - val_accuracy: 0.9333
Epoch 8/40
6/6 [==============================] - 70s 12s/step - loss: 0.3814 - ac
curacy: 0.8539 - val_loss: 0.0520 - val_accuracy: 1.0000
Epoch 9/40
6/6 [==============================] - 69s 12s/step - loss: 0.1755 - ac
curacy: 0.9438 - val_loss: 0.0679 - val_accuracy: 1.0000
Epoch 10/40
6/6 [==============================] - 80s 13s/step - loss: 0.2149 - ac
curacy: 0.8989 - val_loss: 0.0458 - val_accuracy: 1.0000
Epoch 11/40
6/6 [==============================] - 69s 11s/step - loss: 0.2524 - ac
curacy: 0.9101 - val_loss: 0.0620 - val_accuracy: 1.0000
Epoch 12/40
6/6 [==============================] - 68s 11s/step - loss: 0.0652 - ac
curacy: 0.9775 - val_loss: 0.0093 - val_accuracy: 1.0000
Epoch 13/40
6/6 [==============================] - 68s 11s/step - loss: 0.0711 - ac
curacy: 0.9775 - val_loss: 0.0300 - val_accuracy: 1.0000
Epoch 14/40
6/6 [==============================] - 69s 12s/step - loss: 0.0456 - ac
curacy: 0.9888 - val_loss: 0.0269 - val_accuracy: 1.0000
Epoch 15/40
6/6 [==============================] - 69s 12s/step - loss: 0.3867 - ac
curacy: 0.8539 - val_loss: 0.0296 - val_accuracy: 1.0000
Epoch 16/40
6/6 [==============================] - 68s 11s/step - loss: 0.1492 - ac
curacy: 0.9663 - val_loss: 0.0950 - val_accuracy: 0.9333
Epoch 17/40
6/6 [==============================] - 74s 12s/step - loss: 0.0872 - ac
curacy: 0.9663 - val_loss: 0.0136 - val_accuracy: 1.0000
Epoch 18/40
6/6 [==============================] - 71s 12s/step - loss: 0.1058 - ac
curacy: 0.9551 - val_loss: 0.0348 - val_accuracy: 1.0000
Epoch 19/40
6/6 [==============================] - 70s 12s/step - loss: 0.1608 - ac
curacy: 0.9222 - val_loss: 0.0276 - val_accuracy: 1.0000
```

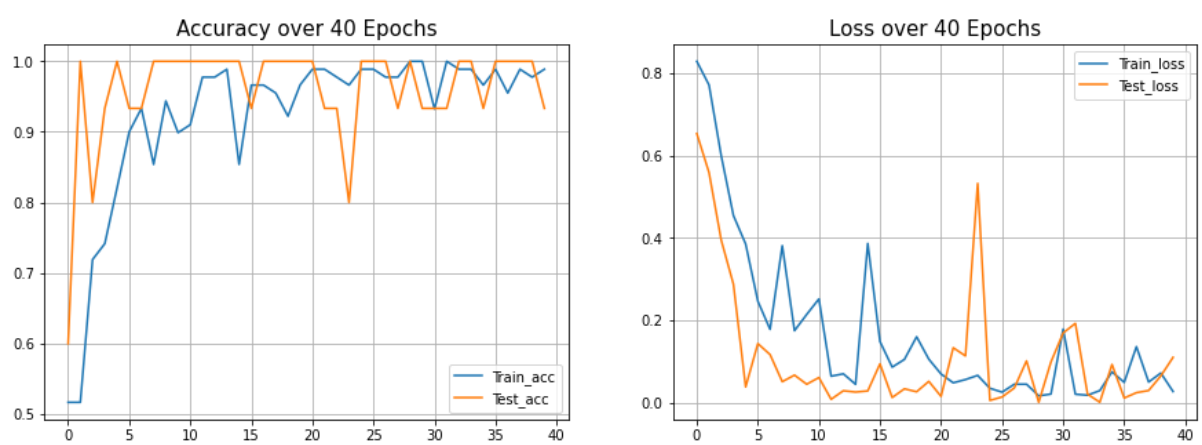
```
Epoch 20/40
6/6 [==============================] - 70s 12s/step - loss: 0.1064 - ac
curacy: 0.9667 - val_loss: 0.0528 - val_accuracy: 1.0000
Epoch 21/40
6/6 [==============================] - 69s 12s/step - loss: 0.0707 - ac
curacy: 0.9888 - val_loss: 0.0164 - val_accuracy: 1.0000
Epoch 22/40
6/6 [==============================] - 69s 12s/step - loss: 0.0493 - ac
curacy: 0.9888 - val_loss: 0.1342 - val_accuracy: 0.9333
Epoch 23/40
6/6 [==============================] - 70s 12s/step - loss: 0.0571 - ac
curacy: 0.9775 - val_loss: 0.1144 - val_accuracy: 0.9333
Epoch 24/40
6/6 [==============================] - 70s 12s/step - loss: 0.0672 - ac
curacy: 0.9663 - val_loss: 0.5321 - val_accuracy: 0.8000
Epoch 25/40
6/6 [==============================] - 71s 12s/step - loss: 0.0360 - ac
curacy: 0.9888 - val_loss: 0.0066 - val_accuracy: 1.0000
Epoch 26/40
6/6 [==============================] - 69s 11s/step - loss: 0.0267 - ac
curacy: 0.9888 - val_loss: 0.0150 - val_accuracy: 1.0000
Epoch 27/40
6/6 [==============================] - 70s 12s/step - loss: 0.0459 - ac
curacy: 0.9775 - val_loss: 0.0367 - val_accuracy: 1.0000
Epoch 28/40
6/6 [==============================] - 70s 12s/step - loss: 0.0459 - ac
curacy: 0.9775 - val_loss: 0.1024 - val_accuracy: 0.9333
Epoch 29/40
6/6 [==============================] - 69s 11s/step - loss: 0.0178 - ac
curacy: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 30/40
6/6 [==============================] - 70s 12s/step - loss: 0.0220 - ac
curacy: 1.0000 - val_loss: 0.0997 - val_accuracy: 0.9333
Epoch 31/40
6/6 [==============================] - 69s 12s/step - loss: 0.1793 - ac
curacy: 0.9326 - val_loss: 0.1700 - val_accuracy: 0.9333
Epoch 32/40
6/6 [==============================] - 75s 12s/step - loss: 0.0217 - ac
curacy: 1.0000 - val_loss: 0.1932 - val_accuracy: 0.9333
Epoch 33/40
6/6 [==============================] - 69s 12s/step - loss: 0.0193 - ac
curacy: 0.9888 - val_loss: 0.0217 - val_accuracy: 1.0000
Epoch 34/40
6/6 [==============================] - 69s 12s/step - loss: 0.0304 - ac
curacy: 0.9888 - val_loss: 0.0022 - val_accuracy: 1.0000
Epoch 35/40
6/6 [==============================] - 70s 12s/step - loss: 0.0758 - ac
curacy: 0.9663 - val_loss: 0.0937 - val_accuracy: 0.9333
Epoch 36/40
6/6 [==============================] - 70s 12s/step - loss: 0.0503 - ac
curacy: 0.9888 - val_loss: 0.0124 - val_accuracy: 1.0000
Epoch 37/40
6/6 [==============================] - 70s 12s/step - loss: 0.1368 - ac
curacy: 0.9551 - val_loss: 0.0254 - val_accuracy: 1.0000
Epoch 38/40
6/6 [==============================] - 69s 12s/step - loss: 0.0511 - ac
curacy: 0.9888 - val_loss: 0.0306 - val_accuracy: 1.0000
```

```
Epoch 39/40
6/6 [==============================] - 79s 13s/step - loss: 0.0730 - ac
curacy: 0.9775 - val_loss: 0.0660 - val_accuracy: 1.0000
Epoch 40/40
6/6 [==============================] - 71s 12s/step - loss: 0.0282 - ac
curacy: 0.9888 - val_loss: 0.1109 - val_accuracy: 0.9333
The execution time is: 3379.2010641098022 seconds
```

In [43]:
```
model_sgd.save('SGD_Binary.h5')
```

[5 points] Plot Accuracy and Loss During Training of SGD+momentum+nesterov optimizer Model

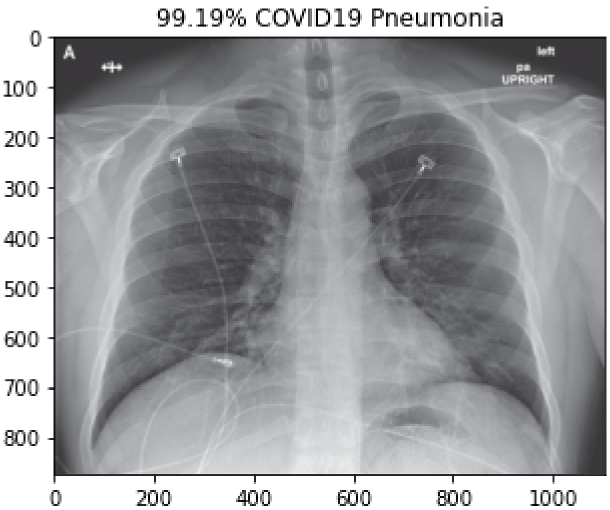
In [44]:
```
import matplotlib.pyplot as plt

def plot_accuracy_loss(output, epochs):
    training_accuracy = output.history['accuracy']
    training_loss = output.history['loss']
    validation_accuracy = output.history['val_accuracy']
    validation_loss = output.history['val_loss']
    plt.figure(figsize=(15, 5))
    plt.subplot(121)
    plt.plot(range(0,NUM_EPOCHS), training_accuracy[:], label='Train_ac
c')
    plt.plot(range(0,NUM_EPOCHS), validation_accuracy[:], label='Test_ac
c')
    plt.title('Accuracy over ' + str(NUM_EPOCHS) + ' Epochs', size=15)
    plt.legend()
    plt.grid(True)
    plt.subplot(122)
    plt.plot(range(0,NUM_EPOCHS), training_loss[:], label='Train_loss')
    plt.plot(range(0,NUM_EPOCHS), validation_loss[:], label='Test_loss')
    plt.title('Loss over ' + str(NUM_EPOCHS) + ' Epochs', size=15)
    plt.legend()
    plt.grid(True)
    plt.show()

plot_accuracy_loss(output, NUM_EPOCHS)
```



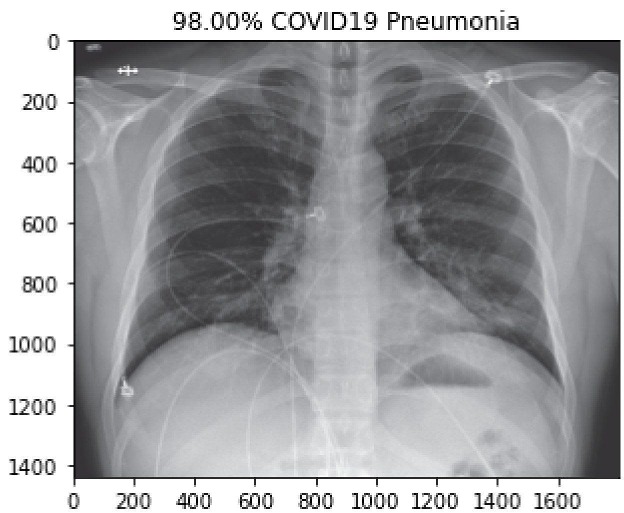
Plot Test Results of SGD+momentum+nesterov optimizer Model

In [11]:
```
import matplotlib.image as mpimg
model_sgd = keras.models.load_model('SGD_Binary.h5')

test_datagen = ImageDataGenerator(rescale=1. / 255)
eval_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=I
MAGE_SIZE,
                                                   batch_size=1,shuffle=F
alse,seed=42,class_mode="binary")
eval_generator.reset()
pred = model_sgd.predict_generator(eval_generator,18,verbose=1)
for index, probability in enumerate(pred):
    image_path = TEST_DIR + "/" +eval_generator.filenames[index]
    image = mpimg.imread(image_path)
    if image.ndim < 3:
        image = np.reshape(image,(image.shape[0],image.shape[1],1))
        image = np.concatenate([image, image, image], 2)
#         print(image.shape)

    pixels = np.array(image)
    plt.imshow(pixels)

    print(eval_generator.filenames[index])
    if probability > 0.5:
        plt.title("%.2f" % (probability[0]*100) + "% Normal")
    else:
        plt.title("%.2f" % ((1-probability[0])*100) + "% COVID19 Pneumon
ia")
    plt.show()
```

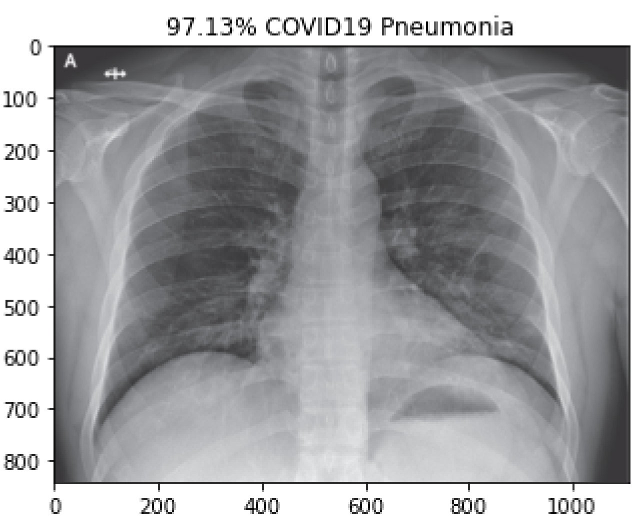
```
Found 18 images belonging to 2 classes.
18/18 [==============================] - 5s 259ms/step
covid/nejmoa2001191_f3-PA.jpeg
```



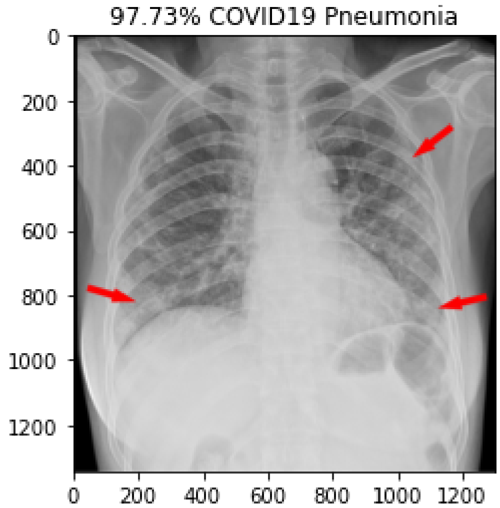

```
covid/nejmoa2001191_f4.jpeg
```



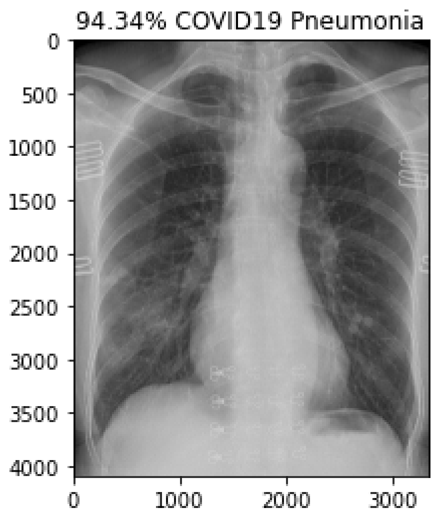

```
covid/nejmoa2001191_f5-PA.jpeg
```



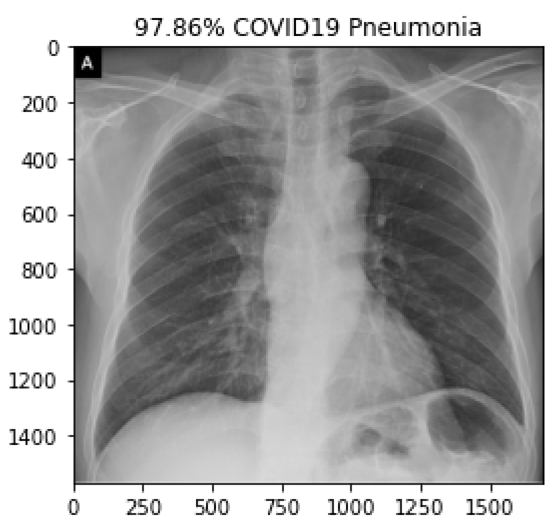

```
covid/radiol.2020200490.fig3.jpeg
```



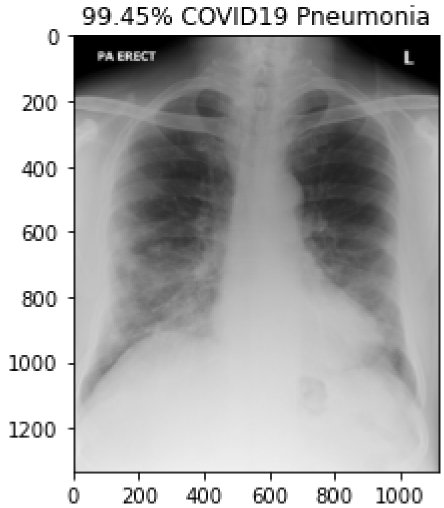
covid/ryct.2020200028.fig1a.jpeg



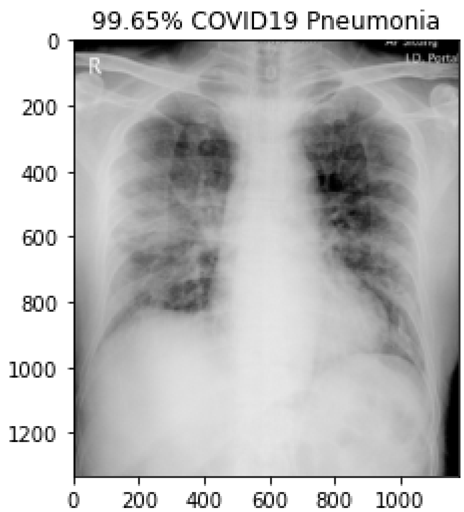
covid/ryct.2020200034.fig2.jpeg



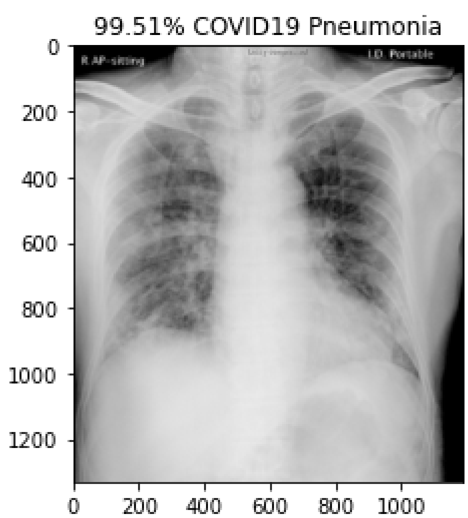
covid/ryct.2020200034.fig5-day0.jpeg



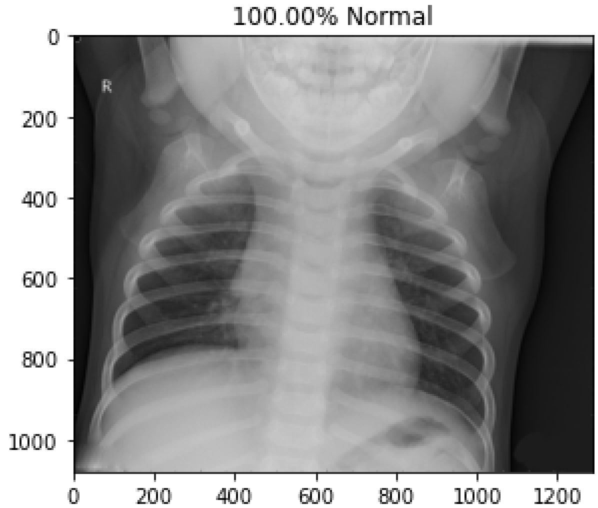
covid/ryct.2020200034.fig5-day4.jpeg



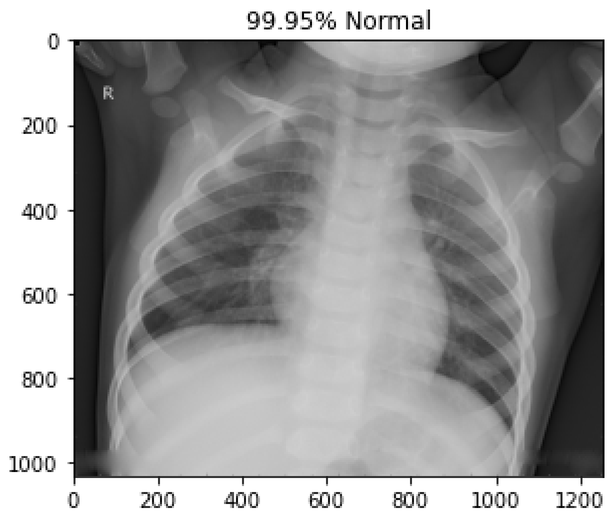
covid/ryct.2020200034.fig5-day7.jpeg



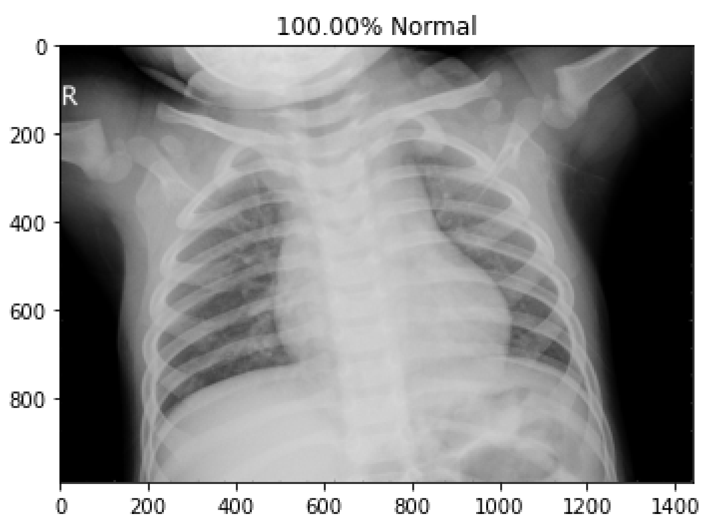
normal/NORMAL2-IM-1385-0001.jpeg



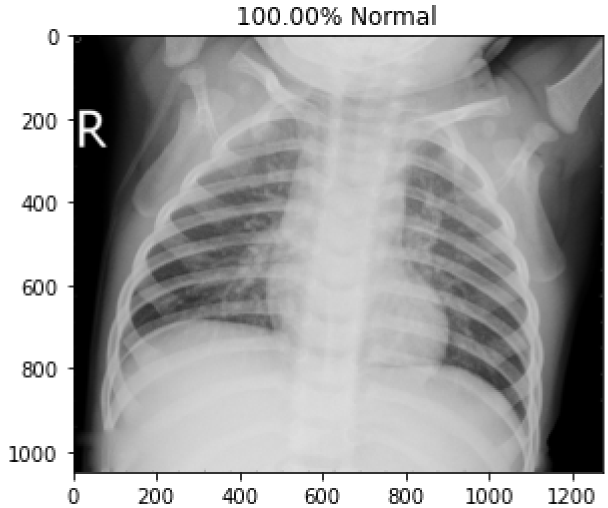
normal/NORMAL2-IM-1396-0001.jpeg



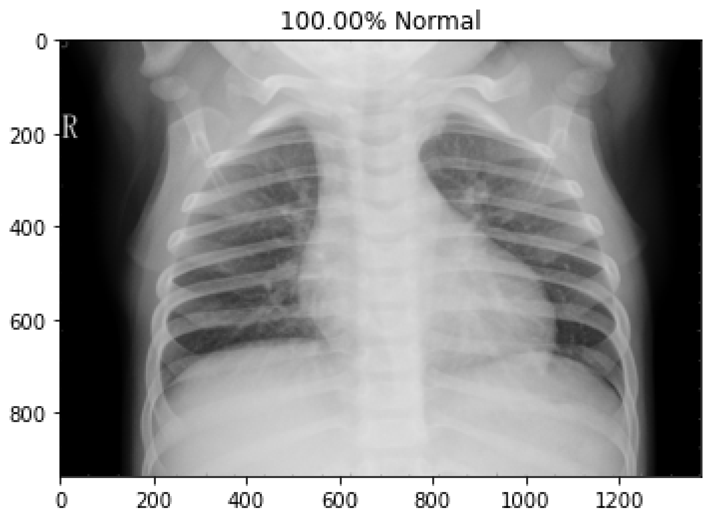
normal/NORMAL2-IM-1400-0001.jpeg



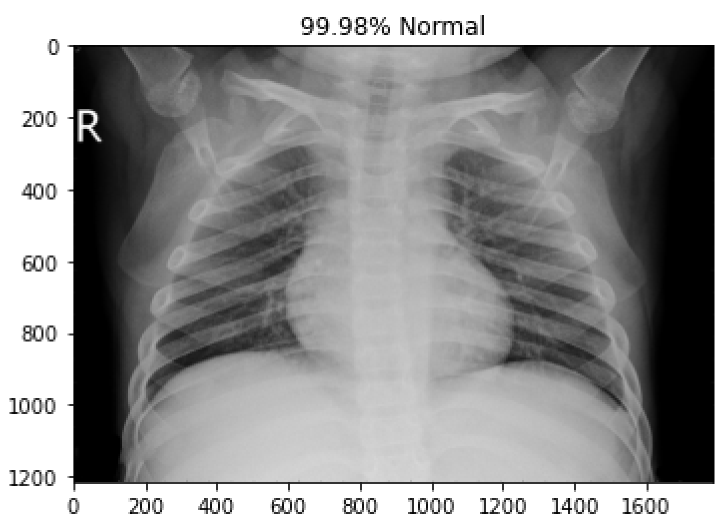
normal/NORMAL2-IM-1401-0001.jpeg



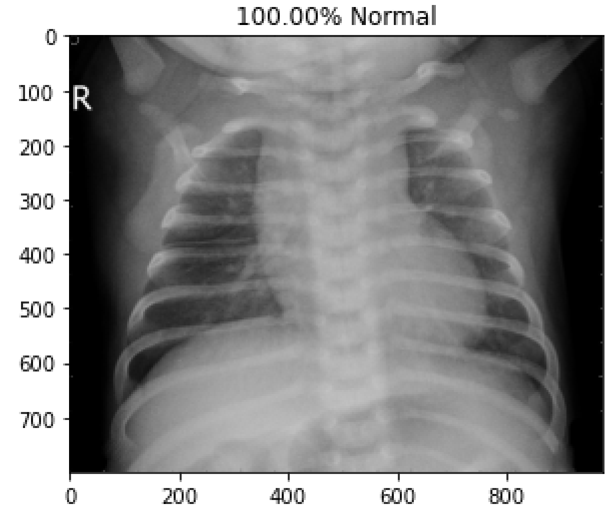
normal/NORMAL2-IM-1406-0001.jpeg



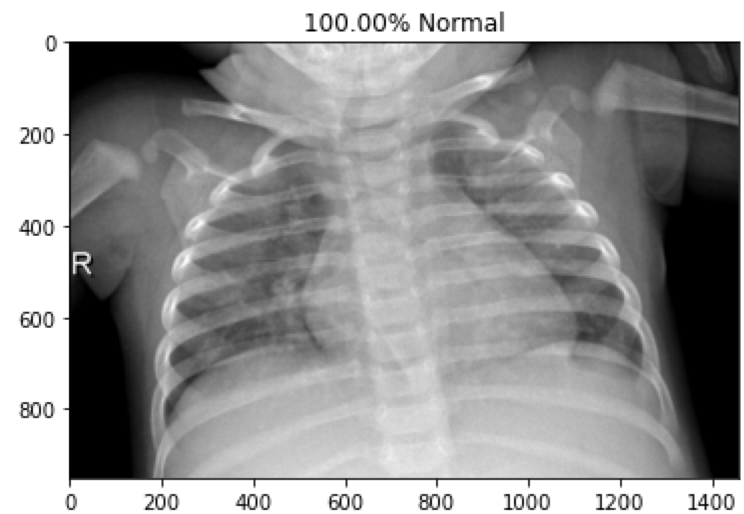
normal/NORMAL2-IM-1412-0001.jpeg



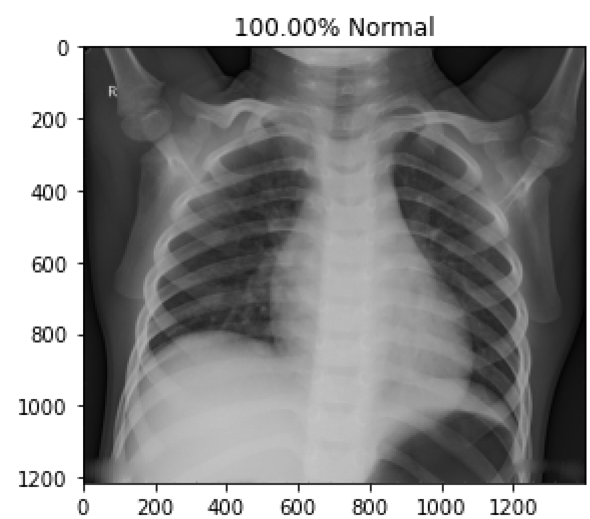
normal/NORMAL2-IM-1419-0001.jpeg



normal/NORMAL2-IM-1422-0001.jpeg



normal/NORMAL2-IM-1423-0001.jpeg



Testing Model of SGD+momentum+nesterov optimizer Model

In [12]:
```
x = model_sgd.evaluate_generator(eval_generator,steps = np.ceil(len(eval
_generator)),
                                 use_multiprocessing = False,verbose = 1,worke
rs=1)
print('Test loss:' , x[0])
print('Test accuracy:',x[1])
```

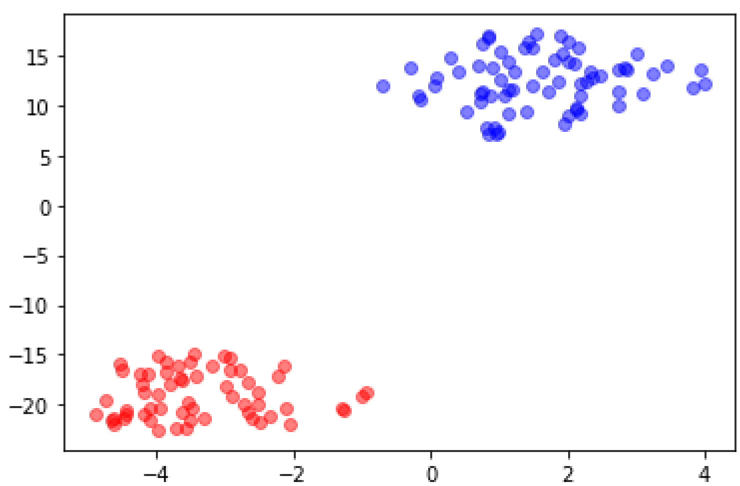
```
18/18 [==============================] - 6s 317ms/step - loss: 0.0097 -
accuracy: 1.0000
Test loss: 0.009725017473101616
Test accuracy: 1.0
```

[10 points] TSNE Plot of SGD+momentum+nesterov optimizer Model

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a widely used technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. After training is complete, extract features from a specific deep layer of your choice, use t-SNE to reduce the dimensionality of your extracted features to 2 dimensions and plot the resulting 2D features.

In [13]:
```
from sklearn.manifold import TSNE
import seaborn as sns
import pandas as pd
from tensorflow import keras

model_sgd = keras.models.load_model('SGD_Binary.h5')

intermediate_layer_model = models.Model(inputs=model_sgd.input,
                                        outputs=model_sgd.get_layer('den
se_feature').output)
tsne_data_generator = test_datagen.flow_from_directory(DATASET_PATH,targ
et_size=IMAGE_SIZE,
                                                  batch_size=1,shuffle=F
alse,seed=42,class_mode="binary")

tsne_data_generator.reset()
activations = intermediate_layer_model.predict_generator(tsne_data_gener
ator, 130, verbose=1)
tsne = TSNE(random_state=42, n_components=2).fit_transform(activations)

for index, tsne in enumerate(tsne):
    if tsne_data_generator.filenames[index][:5] == 'covid':
        plt.scatter(tsne[0], tsne[1], color = 'r', alpha=0.5)
    else:
        plt.scatter(tsne[0], tsne[1], color = 'b', alpha=0.5)
plt.show()
```

```
Found 130 images belonging to 2 classes.
130/130 [==============================] - 37s 284ms/step
```



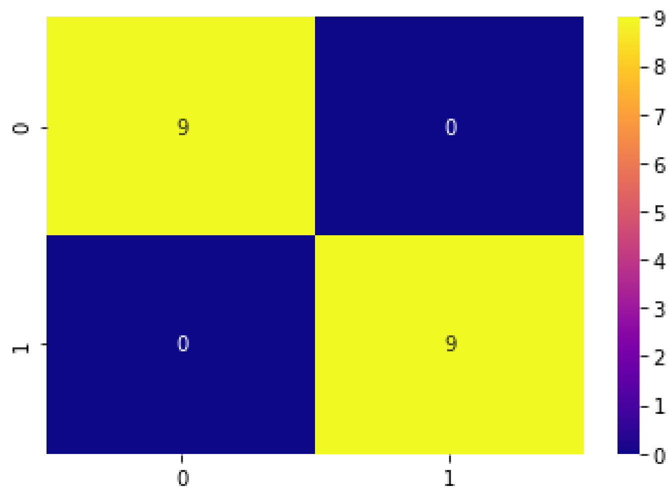
Classification Report and Confusion Matrix of SGD+momentum+nesterov optimizer Model

In [61]:
```
train_batches.class_indices
```

Out[61]:
```
{'covid': 0, 'normal': 1}
```

In [63]:
```
from sklearn.metrics import classification_report, confusion_matrix
import os

model_sgd = tf.keras.models.load_model('SGD_Binary.h5')

y_actual = []
y_test = []

for i in os.listdir('Covid_Data_GradientCrescent/two/test/covid/'):
    y_actual.append(0)

for i in os.listdir('Covid_Data_GradientCrescent/two/test/normal/'):
    y_actual.append(1)

y_actual = np.array(y_actual)
y_test = np.array((model_sgd.predict_generator(eval_generator) > 0.5).as
type("int32"))

cm = confusion_matrix(y_actual, y_test)
sns.heatmap(cm, cmap='plasma', annot=True)

cr = classification_report(y_actual, y_test, target_names=['covid', 'nor
mal'])
print(cr)
```

```
              precision    recall  f1-score   support

       covid       1.00      1.00      1.00         9
      normal       1.00      1.00      1.00         9

    accuracy                           1.00        18
   macro avg       1.00      1.00      1.00        18
weighted avg       1.00      1.00      1.00        18
```



----x----x----x----x----x----x----x----x----x----x----x----x----x----x----x----x----x----x---x---x---

In [1]:
```
import os

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

os.environ['OMP_NUM_THREADS'] = '1'
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
tf.__version__
```

Out[1]:
```
'2.3.0'
```

Load Image Data

In [2]:
```
DATA_LIST = os.listdir('Covid_Data_GradientCrescent/two/train')
DATASET_PATH  = 'Covid_Data_GradientCrescent/two/train'
TEST_DIR =  'Covid_Data_GradientCrescent/two/test'
IMAGE_SIZE    = (224, 224)
NUM_CLASSES   = len(DATA_LIST)
BATCH_SIZE    = 10  # try reducing batch size or freeze more layers if y
our GPU runs out of memory
NUM_EPOCHS    = 40
LEARNING_RATE = 0.001 # start off with high rate first 0.001 and experim
ent with reducing it gradually
```

Generate Training and Validation Batches

In [3]:
```
train_datagen = ImageDataGenerator(rescale=1./255,rotation_range=50,feat
urewise_center = True,
                                   featurewise_std_normalization = True,
width_shift_range=0.2,
                                   height_shift_range=0.2,shear_range=0.
25,zoom_range=0.1,
                                   zca_whitening = True,channel_shift_ra
nge = 20,
                                   horizontal_flip = True,vertical_flip
= True,
                                   validation_split = 0.2,fill_mode='con
stant')

train_batches = train_datagen.flow_from_directory(DATASET_PATH,target_si
ze=IMAGE_SIZE,
                                                  shuffle=True,batch_siz
e=BATCH_SIZE,
                                                  subset = "training",se
ed=42,
                                                  class_mode="binary")

valid_batches = train_datagen.flow_from_directory(DATASET_PATH,target_si
ze=IMAGE_SIZE,
                                                  shuffle=True,batch_siz
e=BATCH_SIZE,
                                                  subset = "validation",
seed=42,
                                                  class_mode="binary")
```

```
Found 104 images belonging to 2 classes.
Found 26 images belonging to 2 classes.
```

```
/Applications/anaconda3/lib/python3.8/site-packages/keras_preprocessin
g/image/image_data_generator.py:342: UserWarning: This ImageDataGenerat
or specifies `zca_whitening` which overrides setting of`featurewise_std
_normalization`.
  warnings.warn('This ImageDataGenerator specifies '
```

Build Model of Adagrad optimizer Model

Hint: Starting from a pre-trained model typically helps performance on a new task, e.g. starting with weights obtained by training on ImageNet.

In [4]:
```
from tensorflow.keras import models, layers, optimizers

vgg16 = tf.keras.applications.VGG16(weights="imagenet", include_top=Fals
e, input_shape=(224,224,3))

model_adagrad = tf.keras.models.Sequential([
    vgg16,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation="relu", name='dense_feature'),
    tf.keras.layers.Dense(1, activation="sigmoid")
])

model_adagrad.compile(loss="binary_crossentropy",
              optimizer=optimizers.Adagrad(lr=LEARNING_RATE),
              metrics=['accuracy'])

model_adagrad.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense_feature (Dense)        (None, 256)               6422784
_________________________________________________________________
dense (Dense)                (None, 1)                 257
=================================================================
Total params: 21,137,729
Trainable params: 21,137,729
Non-trainable params: 0
_________________________________________________________________
```

Train Model of Adagrad optimizer Model

In [5]:
```
#FIT MODEL
print(len(train_batches))
print(len(valid_batches))

STEP_SIZE_TRAIN=train_batches.n//train_batches.batch_size
STEP_SIZE_VALID=valid_batches.n//valid_batches.batch_size

import time
start_time = time.time()

output = model_adagrad.fit(train_batches,
                 validation_data = valid_batches,
                 validation_steps = STEP_SIZE_VALID,
                 batch_size = BATCH_SIZE,
                 steps_per_epoch =STEP_SIZE_TRAIN,
                 epochs= NUM_EPOCHS)

end_time = time.time()
print(f"The execution time is: {end_time-start_time} seconds")
```

```
11
3

/Applications/anaconda3/lib/python3.8/site-packages/keras_preprocessin
g/image/image_data_generator.py:720: UserWarning: This ImageDataGenerat
or specifies `featurewise_center`, but it hasn't been fit on any traini
ng data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
/Applications/anaconda3/lib/python3.8/site-packages/keras_preprocessin
g/image/image_data_generator.py:739: UserWarning: This ImageDataGenerat
or specifies `zca_whitening`, but it hasn't been fit on any training da
ta. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
Epoch 1/40
10/10 [==============================] - 108s 11s/step - loss: 0.9287 -
accuracy: 0.5319 - val_loss: 0.7551 - val_accuracy: 0.5500
Epoch 2/40
10/10 [==============================] - 99s 10s/step - loss: 0.6246 -
accuracy: 0.6596 - val_loss: 0.5474 - val_accuracy: 0.5500
Epoch 3/40
10/10 [==============================] - 107s 11s/step - loss: 0.7880 -
accuracy: 0.5532 - val_loss: 0.5294 - val_accuracy: 0.9000
Epoch 4/40
10/10 [==============================] - 127s 13s/step - loss: 0.5703 -
accuracy: 0.6702 - val_loss: 0.3300 - val_accuracy: 0.9000
Epoch 5/40
10/10 [==============================] - 101s 10s/step - loss: 0.4747 -
accuracy: 0.8298 - val_loss: 0.5115 - val_accuracy: 0.7000
Epoch 6/40
10/10 [==============================] - 96s 10s/step - loss: 0.3880 -
accuracy: 0.8617 - val_loss: 0.1874 - val_accuracy: 0.9500
Epoch 7/40
10/10 [==============================] - 93s 9s/step - loss: 0.3600 - a
ccuracy: 0.8298 - val_loss: 0.1263 - val_accuracy: 1.0000
Epoch 8/40
10/10 [==============================] - 96s 10s/step - loss: 0.2435 -
accuracy: 0.9255 - val_loss: 0.1143 - val_accuracy: 0.9500
Epoch 9/40
10/10 [==============================] - 117s 12s/step - loss: 0.6510 -
accuracy: 0.7553 - val_loss: 0.2546 - val_accuracy: 0.9000
Epoch 10/40
10/10 [==============================] - 115s 11s/step - loss: 0.2105 -
accuracy: 0.9468 - val_loss: 0.1700 - val_accuracy: 0.9000
Epoch 11/40
10/10 [==============================] - 121s 12s/step - loss: 0.1273 -
accuracy: 0.9574 - val_loss: 0.0727 - val_accuracy: 0.9500
Epoch 12/40
10/10 [==============================] - 89s 9s/step - loss: 0.1855 - a
ccuracy: 0.9574 - val_loss: 0.1917 - val_accuracy: 0.9500
Epoch 13/40
10/10 [==============================] - 90s 9s/step - loss: 0.3292 - a
ccuracy: 0.8404 - val_loss: 0.2773 - val_accuracy: 0.9500
Epoch 14/40
10/10 [==============================] - 89s 9s/step - loss: 0.1635 - a
ccuracy: 0.9681 - val_loss: 0.0701 - val_accuracy: 1.0000
Epoch 15/40
10/10 [==============================] - 95s 9s/step - loss: 0.1439 - a
ccuracy: 0.9574 - val_loss: 0.0577 - val_accuracy: 0.9500
Epoch 16/40
10/10 [==============================] - 79s 8s/step - loss: 0.0990 - a
ccuracy: 0.9574 - val_loss: 0.1016 - val_accuracy: 0.9500
Epoch 17/40
10/10 [==============================] - 79s 8s/step - loss: 0.2250 - a
ccuracy: 0.9043 - val_loss: 0.0573 - val_accuracy: 1.0000
Epoch 18/40
10/10 [==============================] - 79s 8s/step - loss: 0.0932 - a
ccuracy: 0.9574 - val_loss: 0.0827 - val_accuracy: 1.0000
Epoch 19/40
10/10 [==============================] - 79s 8s/step - loss: 0.0658 - a
ccuracy: 0.9787 - val_loss: 0.1257 - val_accuracy: 0.9500
```

```
Epoch 20/40
10/10 [==============================] - 79s 8s/step - loss: 0.0897 - a
ccuracy: 0.9681 - val_loss: 0.1609 - val_accuracy: 0.9000
Epoch 21/40
10/10 [==============================] - 86s 9s/step - loss: 0.0630 - a
ccuracy: 0.9787 - val_loss: 0.1416 - val_accuracy: 0.9000
Epoch 22/40
10/10 [==============================] - 79s 8s/step - loss: 0.0581 - a
ccuracy: 0.9894 - val_loss: 0.0384 - val_accuracy: 1.0000
Epoch 23/40
10/10 [==============================] - 90s 9s/step - loss: 0.1301 - a
ccuracy: 0.9468 - val_loss: 0.0203 - val_accuracy: 1.0000
Epoch 24/40
10/10 [==============================] - 90s 9s/step - loss: 0.0873 - a
ccuracy: 0.9681 - val_loss: 0.0287 - val_accuracy: 1.0000
Epoch 25/40
10/10 [==============================] - 103s 10s/step - loss: 0.1192 -
accuracy: 0.9600 - val_loss: 0.0227 - val_accuracy: 1.0000
Epoch 26/40
10/10 [==============================] - 91s 9s/step - loss: 0.0580 - a
ccuracy: 0.9787 - val_loss: 0.0965 - val_accuracy: 0.9500
Epoch 27/40
10/10 [==============================] - 90s 9s/step - loss: 0.2541 - a
ccuracy: 0.8936 - val_loss: 0.8591 - val_accuracy: 0.7500
Epoch 28/40
10/10 [==============================] - 85s 9s/step - loss: 0.1469 - a
ccuracy: 0.9468 - val_loss: 0.0341 - val_accuracy: 1.0000
Epoch 29/40
10/10 [==============================] - 99s 10s/step - loss: 0.0735 -
accuracy: 0.9681 - val_loss: 0.0393 - val_accuracy: 1.0000
Epoch 30/40
10/10 [==============================] - 83s 8s/step - loss: 0.0392 - a
ccuracy: 0.9894 - val_loss: 0.0437 - val_accuracy: 1.0000
Epoch 31/40
10/10 [==============================] - 84s 8s/step - loss: 0.0257 - a
ccuracy: 1.0000 - val_loss: 0.0305 - val_accuracy: 1.0000
Epoch 32/40
10/10 [==============================] - 84s 8s/step - loss: 0.1088 - a
ccuracy: 0.9681 - val_loss: 0.0130 - val_accuracy: 1.0000
Epoch 33/40
10/10 [==============================] - 91s 9s/step - loss: 0.0434 - a
ccuracy: 0.9894 - val_loss: 0.1820 - val_accuracy: 0.9000
Epoch 34/40
10/10 [==============================] - 95s 9s/step - loss: 0.0606 - a
ccuracy: 0.9894 - val_loss: 0.0492 - val_accuracy: 1.0000
Epoch 35/40
10/10 [==============================] - 97s 10s/step - loss: 0.1048 -
accuracy: 0.9574 - val_loss: 0.3205 - val_accuracy: 0.9000
Epoch 36/40
10/10 [==============================] - 78s 8s/step - loss: 0.1522 - a
ccuracy: 0.9149 - val_loss: 0.5103 - val_accuracy: 0.9000
Epoch 37/40
10/10 [==============================] - 79s 8s/step - loss: 0.2720 - a
ccuracy: 0.8936 - val_loss: 1.2157 - val_accuracy: 0.5000
Epoch 38/40
10/10 [==============================] - 78s 8s/step - loss: 0.2525 - a
ccuracy: 0.9149 - val_loss: 0.1053 - val_accuracy: 0.9500
```

```
Epoch 39/40
10/10 [==============================] - 79s 8s/step - loss: 0.0512 - a
ccuracy: 0.9787 - val_loss: 0.0361 - val_accuracy: 1.0000
Epoch 40/40
10/10 [==============================] - 85s 8s/step - loss: 0.0496 - a
ccuracy: 0.9894 - val_loss: 0.1277 - val_accuracy: 0.9500
The execution time is: 4091.3226947784424 seconds
```

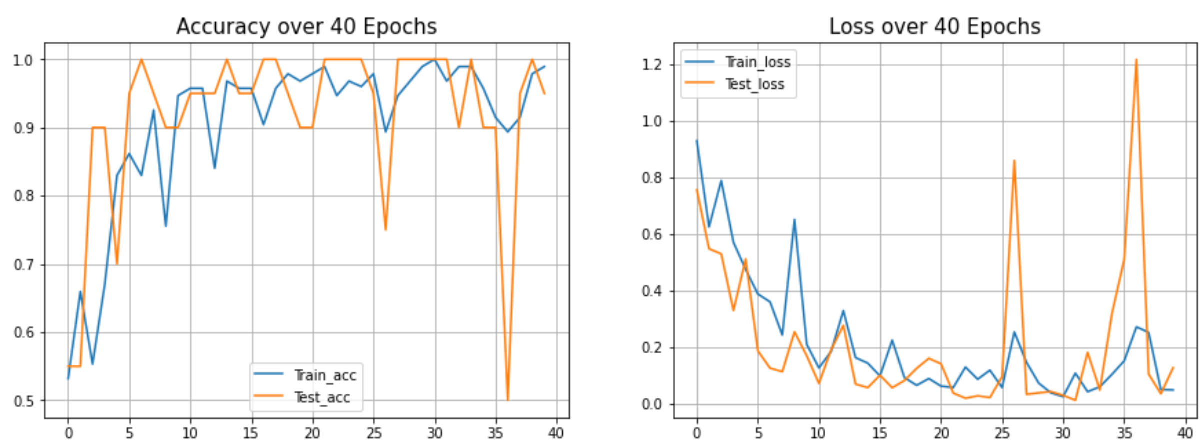
In [21]:

```
model_adagrad.save('Adagrad_Binary.h5')
```

Plot Accuracy and Loss During Training of Adagrad optimizer Model

In [7]:

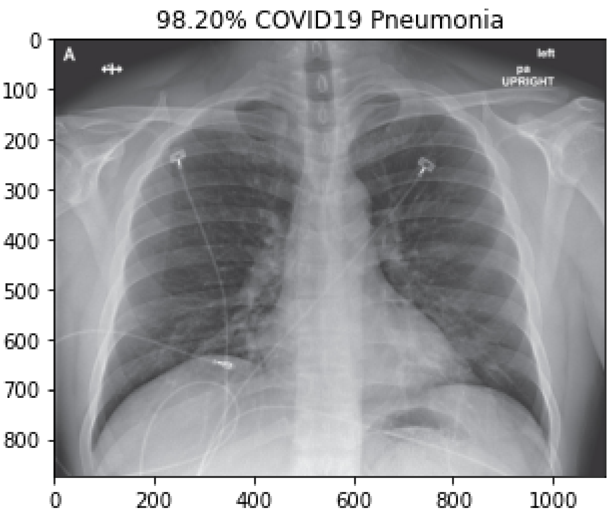
```
import matplotlib.pyplot as plt

def plot_accuracy_loss(output, epochs):
    training_accuracy = output.history['accuracy']
    training_loss = output.history['loss']
    validation_accuracy = output.history['val_accuracy']
    validation_loss = output.history['val_loss']
    plt.figure(figsize=(15, 5))
    plt.subplot(121)
    plt.plot(range(0,NUM_EPOCHS), training_accuracy[:], label='Train_ac
c')
    plt.plot(range(0,NUM_EPOCHS), validation_accuracy[:], label='Test_ac
c')
    plt.title('Accuracy over ' + str(NUM_EPOCHS) + ' Epochs', size=15)
    plt.legend()
    plt.grid(True)
    plt.subplot(122)
    plt.plot(range(0,NUM_EPOCHS), training_loss[:], label='Train_loss')
    plt.plot(range(0,NUM_EPOCHS), validation_loss[:], label='Test_loss')
    plt.title('Loss over ' + str(NUM_EPOCHS) + ' Epochs', size=15)
    plt.legend()
    plt.grid(True)
    plt.show()

plot_accuracy_loss(output, NUM_EPOCHS)
```



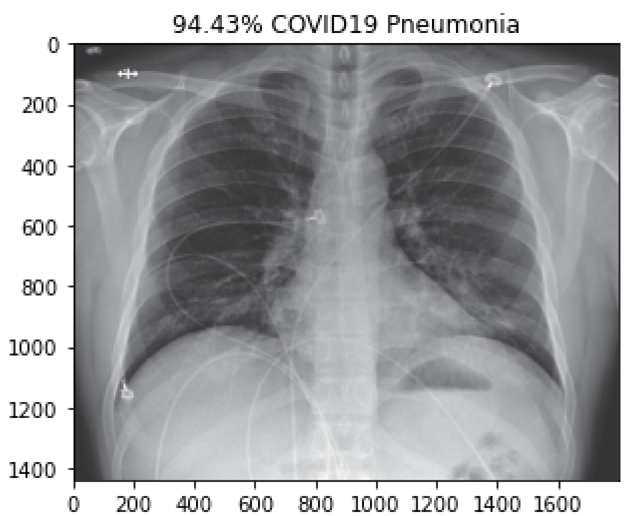
Plot Test Results of Adagrad optimizer Model

In [14]:
```
import matplotlib.image as mpimg
model_adagrad = keras.models.load_model('Adagrad_Binary.h5')

test_datagen = ImageDataGenerator(rescale=1. / 255)
eval_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=I
MAGE_SIZE,
                                                  batch_size=1,shuffle=F
alse,seed=42,class_mode="binary")
eval_generator.reset()
pred = model_adagrad.predict_generator(eval_generator,18,verbose=1)
for index, probability in enumerate(pred):
    image_path = TEST_DIR + "/" +eval_generator.filenames[index]
    image = mpimg.imread(image_path)
    if image.ndim < 3:
        image = np.reshape(image,(image.shape[0],image.shape[1],1))
        image = np.concatenate([image, image, image], 2)
#         print(image.shape)

    pixels = np.array(image)
    plt.imshow(pixels)

    print(eval_generator.filenames[index])
    if probability > 0.5:
        plt.title("%.2f" % (probability[0]*100) + "% Normal")
    else:
        plt.title("%.2f" % ((1-probability[0])*100) + "% COVID19 Pneumon
ia")
    plt.show()
```

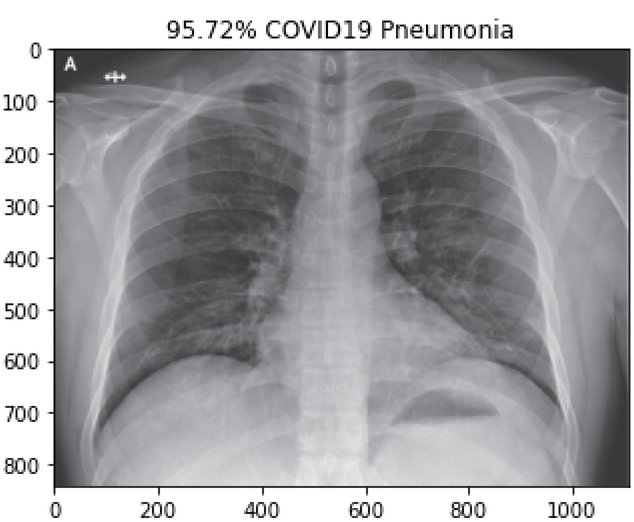
```
Found 18 images belonging to 2 classes.
18/18 [==============================] - 7s 375ms/step
covid/nejmoa2001191_f3-PA.jpeg
```



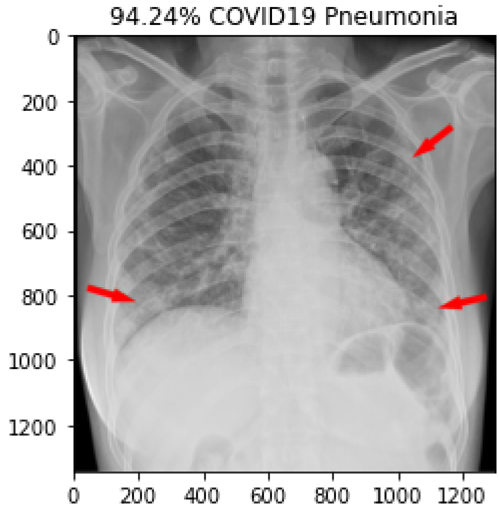

```
covid/nejmoa2001191_f4.jpeg
```



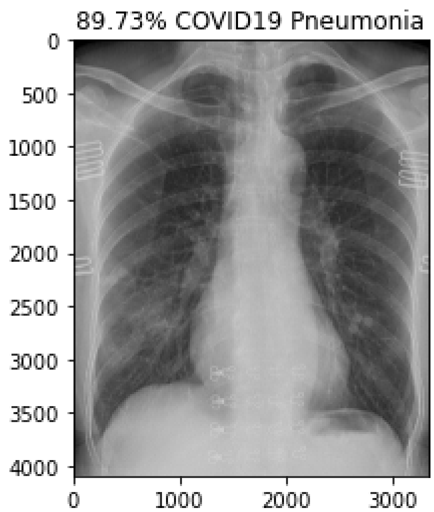

```
covid/nejmoa2001191_f5-PA.jpeg
```



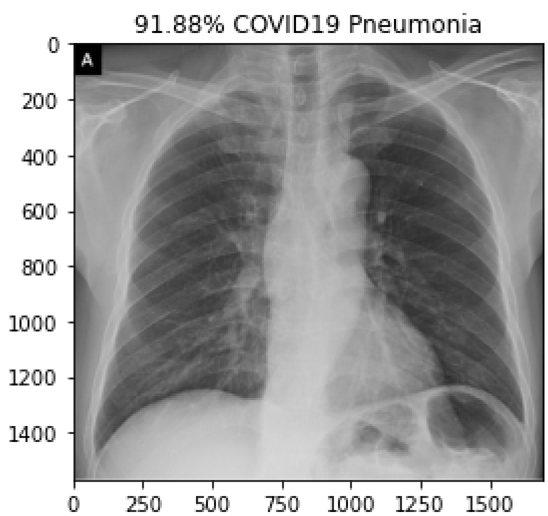

```
covid/radiol.2020200490.fig3.jpeg
```



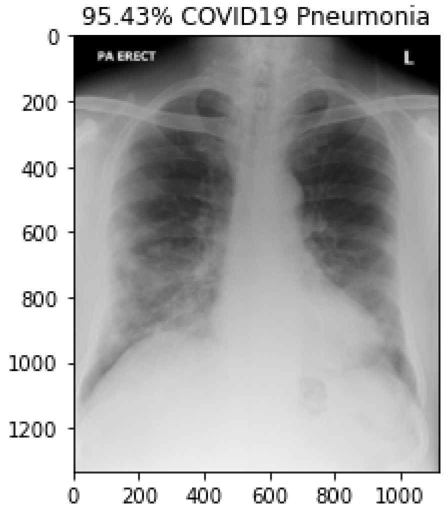
covid/ryct.2020200028.fig1a.jpeg



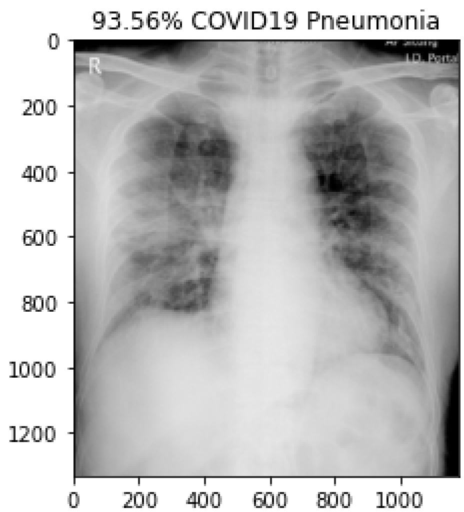
covid/ryct.2020200034.fig2.jpeg



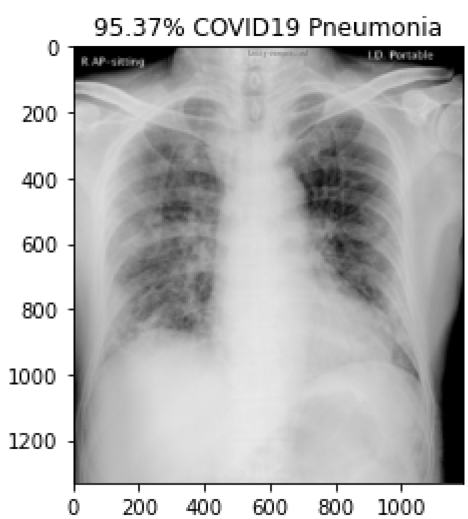
covid/ryct.2020200034.fig5-day0.jpeg



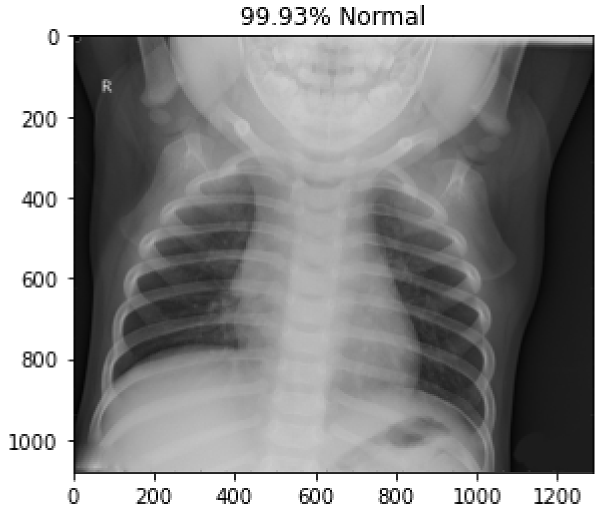
covid/ryct.2020200034.fig5-day4.jpeg



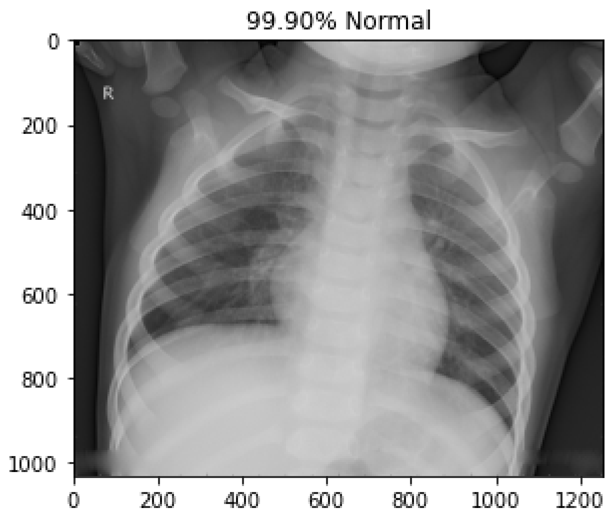
covid/ryct.2020200034.fig5-day7.jpeg



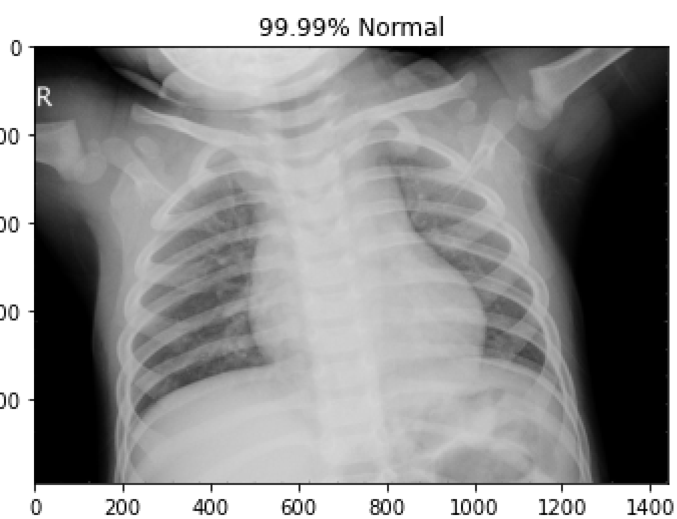
normal/NORMAL2-IM-1385-0001.jpeg



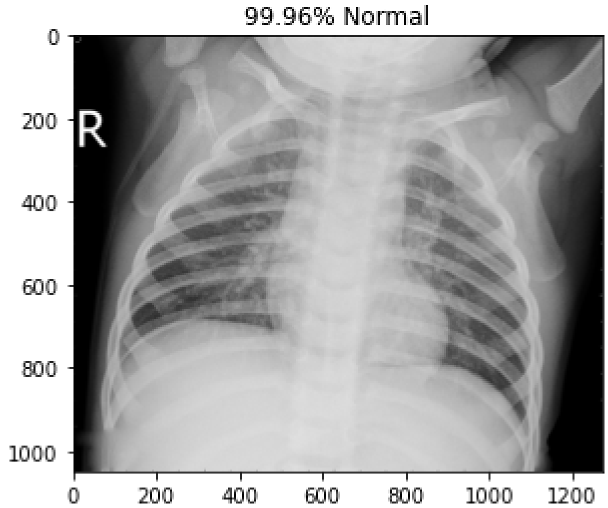
normal/NORMAL2-IM-1396-0001.jpeg



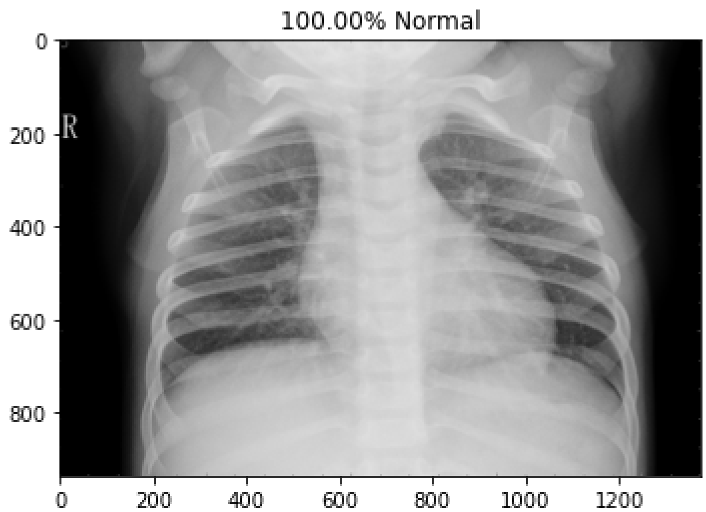
normal/NORMAL2-IM-1400-0001.jpeg



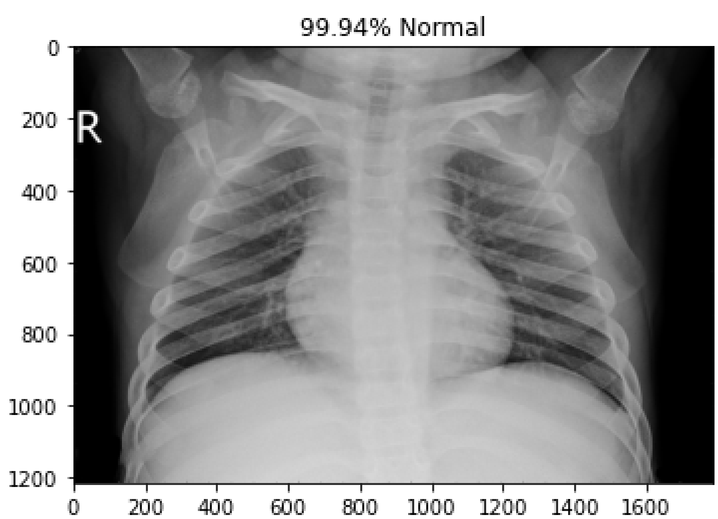
normal/NORMAL2-IM-1401-0001.jpeg



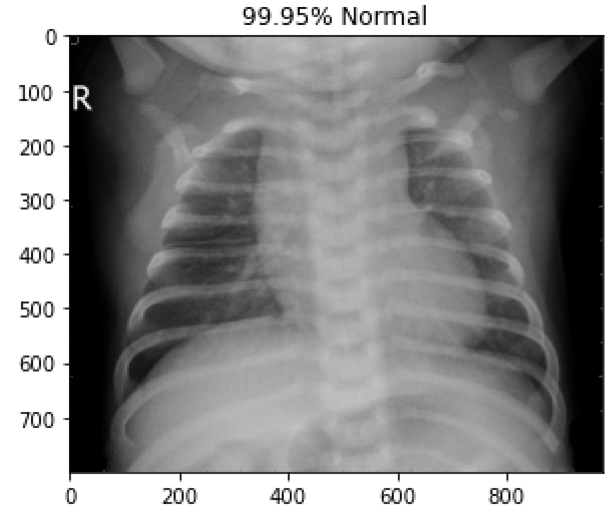
normal/NORMAL2-IM-1406-0001.jpeg



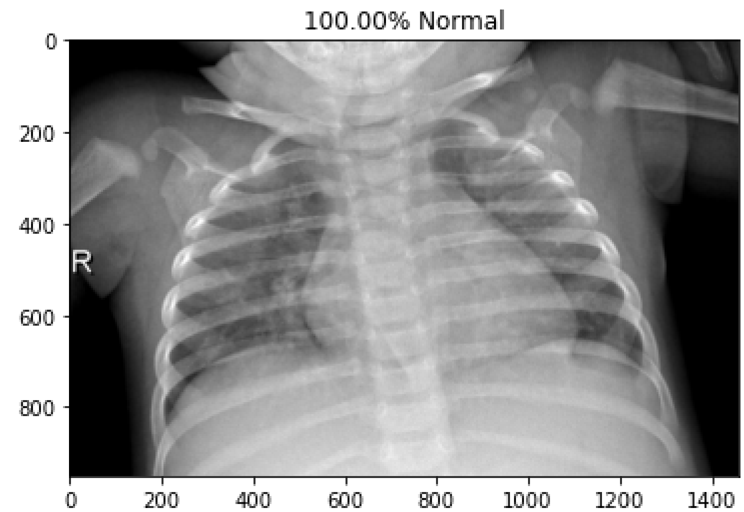
normal/NORMAL2-IM-1412-0001.jpeg



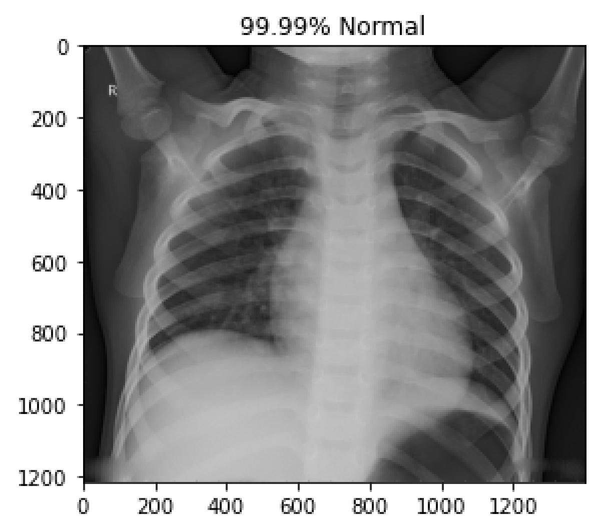
normal/NORMAL2-IM-1419-0001.jpeg




```
normal/NORMAL2-IM-1422-0001.jpeg
```




```
normal/NORMAL2-IM-1423-0001.jpeg
```



Testing Model of Adagrad optimizer Model

In [15]:
```
x = model_adagrad.evaluate_generator(eval_generator,steps = np.ceil(len(
eval_generator)),
                                   use_multiprocessing = False,verbose = 1,worke
rs=1)
print('Test loss:' , x[0])
print('Test accuracy:',x[1])
```

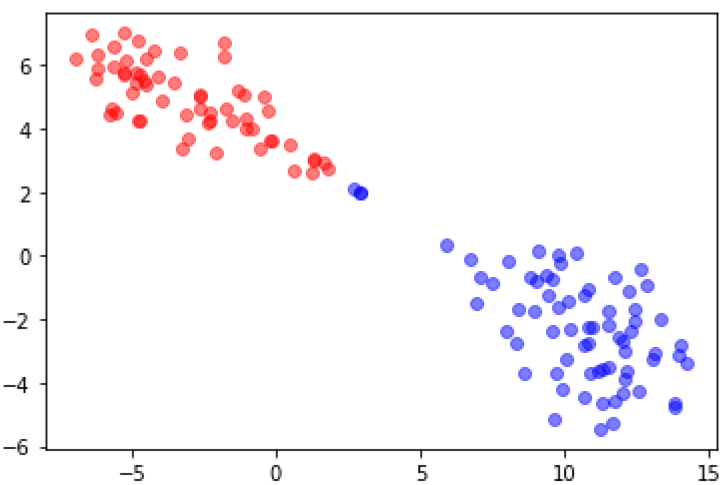
```
18/18 [==============================] - 9s 514ms/step - loss: 0.0298 -
accuracy: 1.0000
Test loss: 0.029754221439361572
Test accuracy: 1.0
```

TSNE Plot of Adagrad optimizer Model

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a widely used technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. After training is complete, extract features from a specific deep layer of your choice, use t-SNE to reduce the dimensionality of your extracted features to 2 dimensions and plot the resulting 2D features.

In [16]:
```
from sklearn.manifold import TSNE
import seaborn as sns
import pandas as pd
from tensorflow import keras

intermediate_layer_model = models.Model(inputs=model_adagrad.input,
                                        outputs=model_adagrad.get_layer(
'dense_feature').output)
tsne_data_generator = test_datagen.flow_from_directory(DATASET_PATH,targ
et_size=IMAGE_SIZE,
                                                        batch_size=1,shuffle=F
alse,seed=42,class_mode="binary")

tsne_data_generator.reset()
activations = intermediate_layer_model.predict_generator(tsne_data_gener
ator, 130, verbose=1)
tsne = TSNE(random_state=42, n_components=2).fit_transform(activations)

for index, tsne in enumerate(tsne):
    if tsne_data_generator.filenames[index][:5] == 'covid':
        plt.scatter(tsne[0], tsne[1], color = 'r', alpha=0.5)
    else:
        plt.scatter(tsne[0], tsne[1], color = 'b', alpha=0.5)
plt.show()
```

```
Found 130 images belonging to 2 classes.
130/130 [==============================] - 35s 272ms/step
```



Classification Report and Confusion Matrix of Adagrad optimizer Model

In [60]:
```
train_batches.class_indices
```

Out[60]:
```
{'covid': 0, 'normal': 1}
```

In [58]:
```
from sklearn.metrics import classification_report, confusion_matrix
import os

y_actual = []
y_test = []

model_adagrad = tf.keras.models.load_model('Adagrad_Binary.h5')

for i in os.listdir('Covid_Data_GradientCrescent/two/test/covid/'):
    y_actual.append(0)

for i in os.listdir('Covid_Data_GradientCrescent/two/test/normal/'):
    y_actual.append(1)


y_actual = np.array(y_actual)
y_test = np.array((model_adagrad.predict_generator(eval_generator) > 0.5
).astype("int32"))

cm = confusion_matrix(y_actual, y_test)
sns.heatmap(cm, cmap='plasma', annot=True)

cr = classification_report(y_actual, y_test, target_names=['covid', 'nor
mal'])
print(cr)
```

```
              precision    recall  f1-score   support

       covid       1.00      1.00      1.00         9
      normal       1.00      1.00      1.00         9

    accuracy                           1.00        18
   macro avg       1.00      1.00      1.00        18
weighted avg       1.00      1.00      1.00        18
```