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]:

```
'2.3.1'
```

Load Image Data

In [2]:

```
DATA_LIST = os.listdir('two/train')
DATASET_PATH  = 'two/train'
TEST_DIR =  'two/test'
IMAGE_SIZE    = (224, 224)
NUM_CLASSES   = len(DATA_LIST)
BATCH_SIZE    = 10  # try reducing batch size or freeze more layers if your GPU runs out of memo
ry
NUM_EPOCHS    = 40
LEARNING_RATE = 0.0005 # start off with high rate first 0.001 and experiment with reducing it gr
adually
```

Generate Training and Validation Batches

In [3]:

```
train_datagen = ImageDataGenerator(rescale=1./255,rotation_range=50,featurewise_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_range = 20,
                                   horizontal_flip = True,vertical_flip = True,
                                   validation_split = 0.2,fill_mode='constant')

train_batches = train_datagen.flow_from_directory(DATASET_PATH,target_size=IMAGE_SIZE,
                                                  shuffle=True,batch_size=BATCH_SIZE,
                                                  subset = "training",seed=42,
                                                  class_mode="binary")

valid_batches = train_datagen.flow_from_directory(DATASET_PATH,target_size=IMAGE_SIZE,
                                                  shuffle=True,batch_size=BATCH_SIZE,
                                                  subset = "validation",seed=42,
                                                  class_mode="binary")
```

```
C:₩Users₩Soo Whan Park₩Anaconda3₩lib₩site-packages₩keras_preprocessing₩image₩image
_data_generator.py:342: UserWarning: This ImageDataGenerator specifies `zca_whiten
ing` which overrides setting of`featurewise_std_normalization`.
  warnings.warn('This ImageDataGenerator specifies '

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

[10 points] Build 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 [6]:

```
vgg16 = tf.keras.applications.VGG16(weights='imagenet', include_top=False,input_shape = (224,22
4,3))
vgg19 = tf.keras.applications.VGG19(include_top=False, weights='imagenet',input_shape=(224,224,
3))
model = tf.keras.models.Sequential([
    vgg16,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu', name='dense_feature'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
vgg16.trainable=False
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688  
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_feature (Dense)        (None, 256)               6422784   
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257       
=================================================================
Total params: 21,137,729
Trainable params: 6,423,041
Non-trainable params: 14,714,688
_________________________________________________________________
```

[5 points] Train Model

In [7]:

```
#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

opt = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
model.compile(optimizer=opt,loss=tf.keras.losses.BinaryCrossentropy(),metrics=['accuracy'])
history = model.fit(train_batches, epochs=NUM_EPOCHS, validation_data=valid_batches,batch_size=5
, steps_per_epoch=STEP_SIZE_TRAIN, validation_steps = STEP_SIZE_VALID)
```

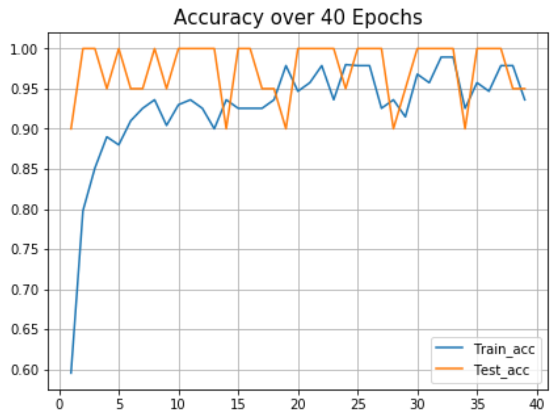
```
11
3
Epoch 1/40
10/10 [==============================] - 11s 1s/step - loss: 1.0780 - accuracy: 0.
6596 - val_loss: 0.5062 - val_accuracy: 0.8000
Epoch 2/40
10/10 [==============================] - 13s 1s/step - loss: 0.3665 - accuracy: 0.
8298 - val_loss: 0.5698 - val_accuracy: 0.7000
Epoch 3/40
10/10 [==============================] - 11s 1s/step - loss: 0.3010 - accuracy: 0.
8936 - val_loss: 0.0846 - val_accuracy: 0.9500
Epoch 4/40
10/10 [==============================] - 11s 1s/step - loss: 0.3280 - accuracy: 0.
8723 - val_loss: 0.1715 - val_accuracy: 0.9000
Epoch 5/40
10/10 [==============================] - 14s 1s/step - loss: 0.1932 - accuracy: 0.
9574 - val_loss: 0.1683 - val_accuracy: 0.9000
Epoch 6/40
10/10 [==============================] - 13s 1s/step - loss: 0.1884 - accuracy: 0.
9000 - val_loss: 0.0277 - val_accuracy: 1.0000
Epoch 7/40
10/10 [==============================] - 13s 1s/step - loss: 0.1729 - accuracy: 0.
9468 - val_loss: 0.1226 - val_accuracy: 0.9500
Epoch 8/40
10/10 [==============================] - 13s 1s/step - loss: 0.4017 - accuracy: 0.
8298 - val_loss: 0.2315 - val_accuracy: 0.9000
Epoch 9/40
10/10 [==============================] - 12s 1s/step - loss: 0.2807 - accuracy: 0.
8830 - val_loss: 0.5228 - val_accuracy: 0.8500
Epoch 10/40
10/10 [==============================] - 12s 1s/step - loss: 0.3957 - accuracy: 0.
8723 - val_loss: 0.2355 - val_accuracy: 0.9500
Epoch 11/40
10/10 [==============================] - 12s 1s/step - loss: 0.1333 - accuracy: 0.
9468 - val_loss: 0.0570 - val_accuracy: 0.9500
Epoch 12/40
10/10 [==============================] - 13s 1s/step - loss: 0.2152 - accuracy: 0.
9255 - val_loss: 0.0577 - val_accuracy: 1.0000
Epoch 13/40
10/10 [==============================] - 14s 1s/step - loss: 0.2735 - accuracy: 0.
8936 - val_loss: 0.1375 - val_accuracy: 0.9500
Epoch 14/40
10/10 [==============================] - 13s 1s/step - loss: 0.2333 - accuracy: 0.
9043 - val_loss: 0.5374 - val_accuracy: 0.8000
Epoch 15/40
10/10 [==============================] - 13s 1s/step - loss: 0.5257 - accuracy: 0.
8191 - val_loss: 0.0100 - val_accuracy: 1.0000
Epoch 16/40
10/10 [==============================] - 13s 1s/step - loss: 0.3143 - accuracy: 0.
8900 - val_loss: 0.2674 - val_accuracy: 0.9500
Epoch 17/40
10/10 [==============================] - 12s 1s/step - loss: 0.1551 - accuracy: 0.
9149 - val_loss: 0.1731 - val_accuracy: 0.9500
Epoch 18/40
10/10 [==============================] - 12s 1s/step - loss: 0.3488 - accuracy: 0.
9255 - val_loss: 0.0660 - val_accuracy: 0.9500
Epoch 19/40
10/10 [==============================] - 13s 1s/step - loss: 0.2506 - accuracy: 0.
8900 - val_loss: 0.0665 - val_accuracy: 1.0000
Epoch 20/40
10/10 [==============================] - 13s 1s/step - loss: 0.2272 - accuracy: 0.
```

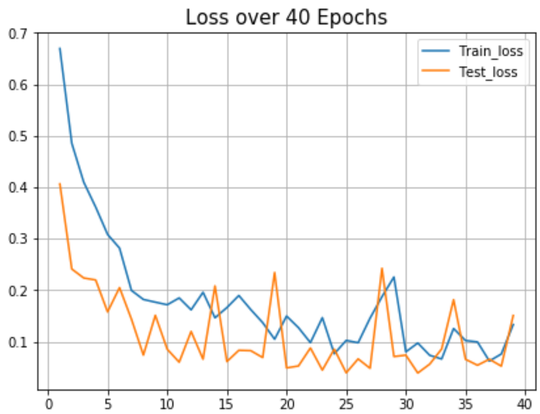
```
9255 - val_loss: 0.4688 - val_accuracy: 0.8500
Epoch 21/40
10/10 [==============================] - 13s 1s/step - loss: 0.2869 - accuracy: 0.
8830 - val_loss: 0.0550 - val_accuracy: 1.0000
Epoch 22/40
10/10 [==============================] - 13s 1s/step - loss: 0.2386 - accuracy: 0.
9149 - val_loss: 0.1267 - val_accuracy: 0.9500
Epoch 23/40
10/10 [==============================] - 13s 1s/step - loss: 0.1224 - accuracy: 0.
9574 - val_loss: 0.0974 - val_accuracy: 0.9500
Epoch 24/40
10/10 [==============================] - 12s 1s/step - loss: 0.0524 - accuracy: 0.
9787 - val_loss: 0.0802 - val_accuracy: 1.0000
Epoch 25/40
10/10 [==============================] - 12s 1s/step - loss: 0.0812 - accuracy: 0.
9681 - val_loss: 0.6811 - val_accuracy: 0.7500
Epoch 26/40
10/10 [==============================] - 12s 1s/step - loss: 0.2156 - accuracy: 0.
9149 - val_loss: 0.0404 - val_accuracy: 1.0000
Epoch 27/40
10/10 [==============================] - 12s 1s/step - loss: 0.1360 - accuracy: 0.
9255 - val_loss: 0.0290 - val_accuracy: 1.0000
Epoch 28/40
10/10 [==============================] - 12s 1s/step - loss: 0.1194 - accuracy: 0.
9681 - val_loss: 0.1862 - val_accuracy: 0.8500
Epoch 29/40
10/10 [==============================] - 13s 1s/step - loss: 0.1604 - accuracy: 0.
9362 - val_loss: 0.0978 - val_accuracy: 0.9500
Epoch 30/40
10/10 [==============================] - 13s 1s/step - loss: 0.0265 - accuracy: 1.
0000 - val_loss: 0.0297 - val_accuracy: 1.0000
Epoch 31/40
10/10 [==============================] - 13s 1s/step - loss: 0.1021 - accuracy: 0.
9681 - val_loss: 0.0171 - val_accuracy: 1.0000
Epoch 32/40
10/10 [==============================] - 12s 1s/step - loss: 0.0856 - accuracy: 0.
9787 - val_loss: 0.0674 - val_accuracy: 1.0000
Epoch 33/40
10/10 [==============================] - 13s 1s/step - loss: 0.0450 - accuracy: 0.
9800 - val_loss: 0.1093 - val_accuracy: 0.9000
Epoch 34/40
10/10 [==============================] - 12s 1s/step - loss: 0.0588 - accuracy: 0.
9681 - val_loss: 0.0937 - val_accuracy: 0.9500
Epoch 35/40
10/10 [==============================] - 12s 1s/step - loss: 0.0621 - accuracy: 0.
9787 - val_loss: 0.0929 - val_accuracy: 0.9500
Epoch 36/40
10/10 [==============================] - 12s 1s/step - loss: 0.0324 - accuracy: 1.
0000 - val_loss: 0.0549 - val_accuracy: 1.0000
Epoch 37/40
10/10 [==============================] - 13s 1s/step - loss: 0.0510 - accuracy: 0.
9787 - val_loss: 0.0170 - val_accuracy: 1.0000
Epoch 38/40
10/10 [==============================] - 12s 1s/step - loss: 0.0402 - accuracy: 0.
9894 - val_loss: 0.0280 - val_accuracy: 1.0000
Epoch 39/40
10/10 [==============================] - 12s 1s/step - loss: 0.0950 - accuracy: 0.
9787 - val_loss: 0.0176 - val_accuracy: 1.0000
Epoch 40/40
10/10 [==============================] - 12s 1s/step - loss: 0.0679 - accuracy: 0.
9800 - val_loss: 0.1045 - val_accuracy: 0.9000
```

[5 points] Plot Accuracy and Loss During Training

In [11]:

```
import matplotlib.pyplot as plt

plt.title('Accuracy over 40 Epochs')
plt.plot(history.history['accuracy'], label='Train_acc')
plt.plot(history.history['val_accuracy'], label = 'Test_acc')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

plt.title('Loss over 40 Epochs')
plt.plot(history.history['loss'], label='Train_loss')
plt.plot(history.history['val_loss'], label = 'Test_loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



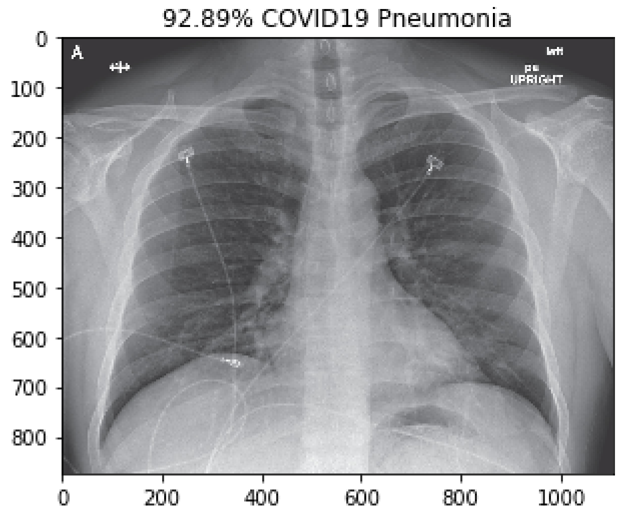


Plot Test Results

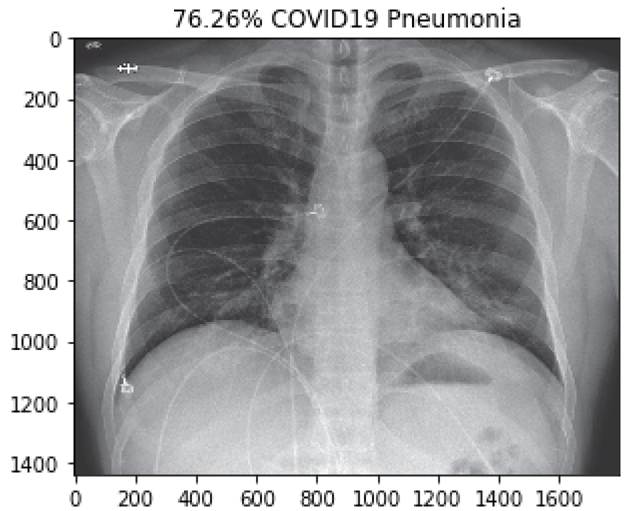
In [12]:

```
import matplotlib.image as mpimg

test_datagen = ImageDataGenerator(rescale=1. / 255)
eval_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=IMAGE_SIZE,
                                                  batch_size=1,shuffle=True,seed=42,class_mode=
"binary")
eval_generator.reset()
pred = model.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 Pneumonia")
    plt.show()
```

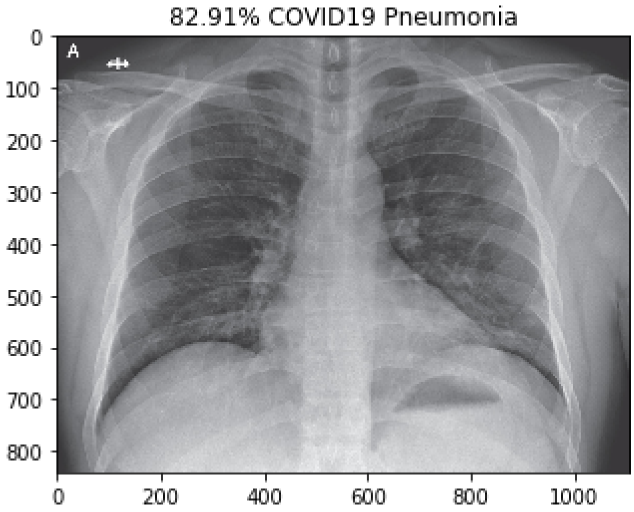
```
Found 18 images belonging to 2 classes.
WARNING:tensorflow:From <ipython-input-12-543347a5fba8>:7: Model.predict_generator
(from tensorflow.python.keras.engine.training) is deprecated and will be removed i
n a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
18/18 [==============================] - 5s 292ms/step
covid₩nejmoa2001191_f3-PA.jpeg
```



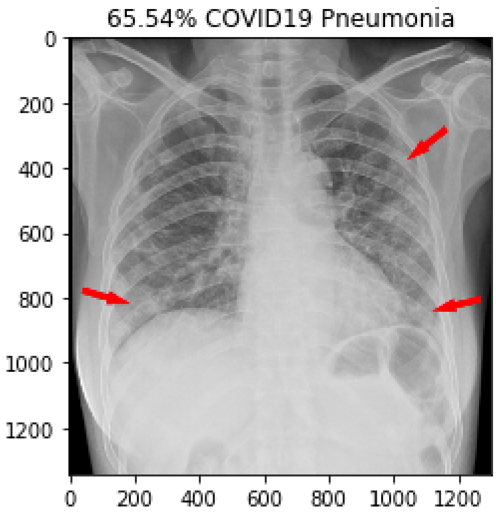
covid₩nejmoa2001191_f4.jpeg



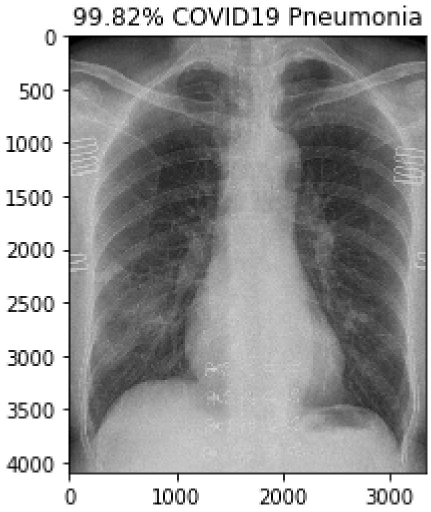
covid₩nejmoa2001191_f5-PA.jpeg



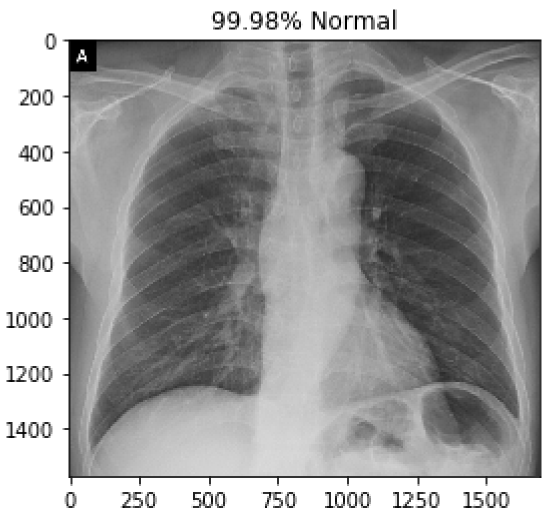
covid₩radiol.2020200490.fig3.jpeg



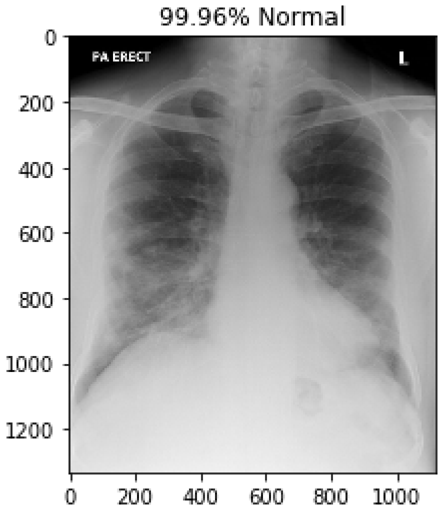
covid₩ryct.2020200028.fig1a.jpeg



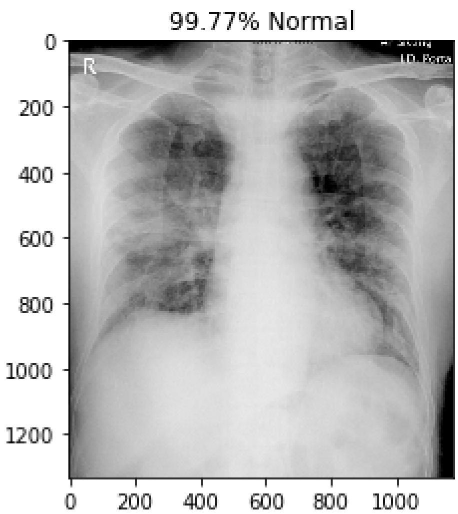
covid₩ryct.2020200034.fig2.jpeg



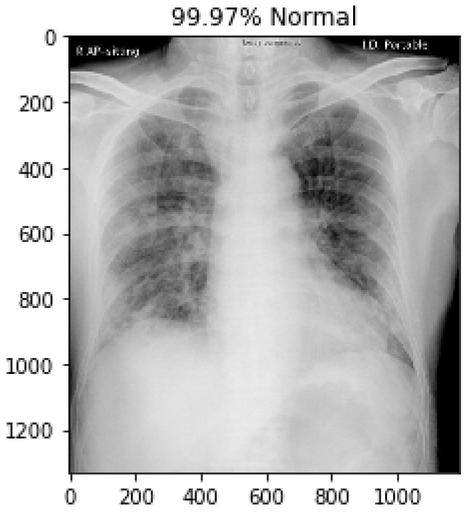
covid₩ryct.2020200034.fig5-day0.jpeg



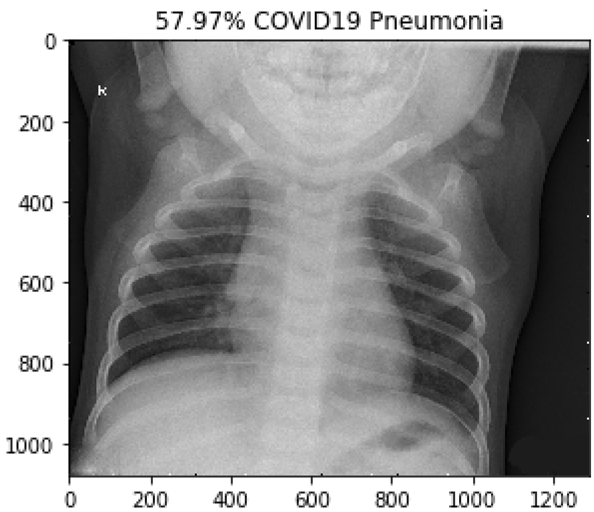
covid₩ryct.2020200034.fig5-day4.jpeg



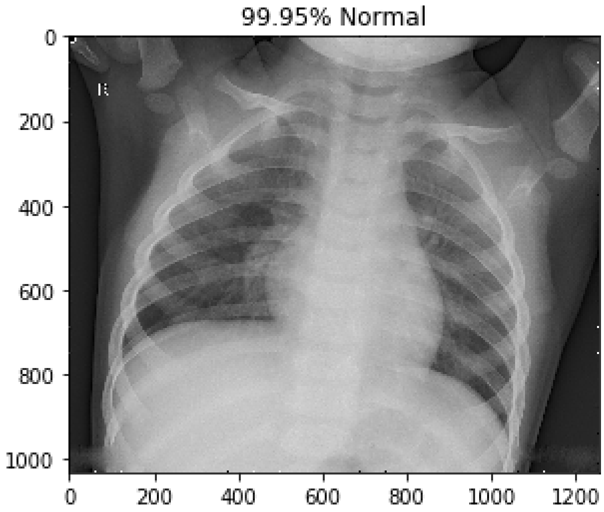
covid₩ryct.2020200034.fig5-day7.jpeg



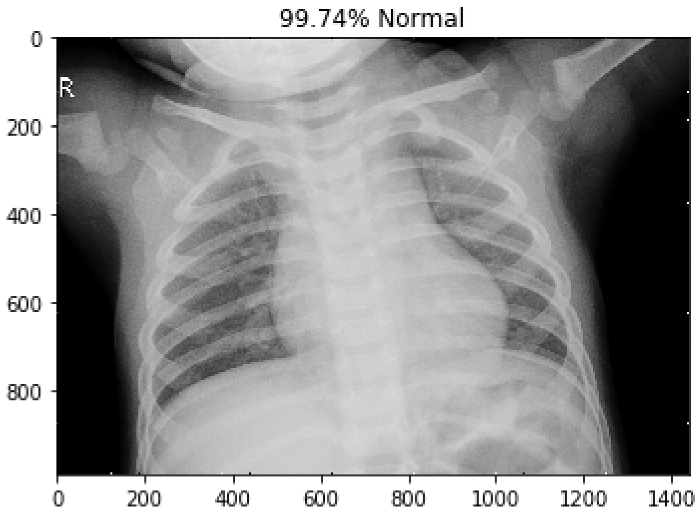
normal₩NORMAL2-IM-1385-0001.jpeg



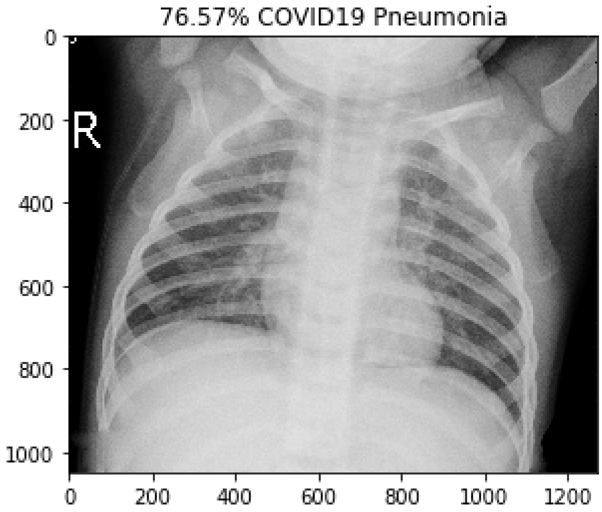
normal₩NORMAL2-IM-1396-0001.jpeg



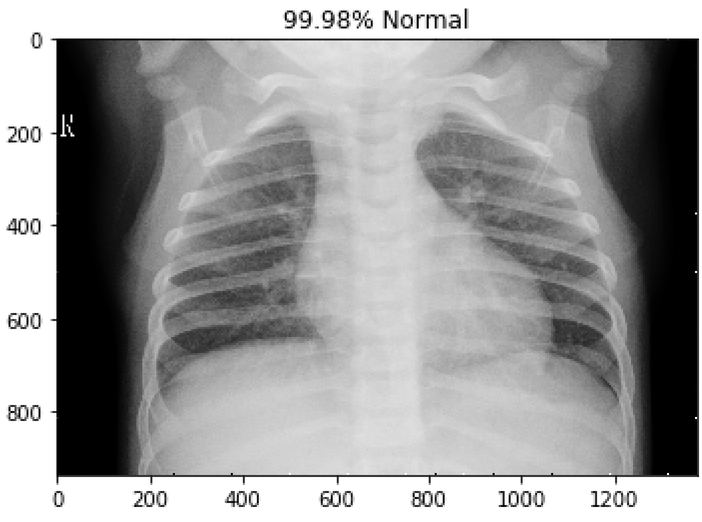
normal₩NORMAL2-IM-1400-0001.jpeg



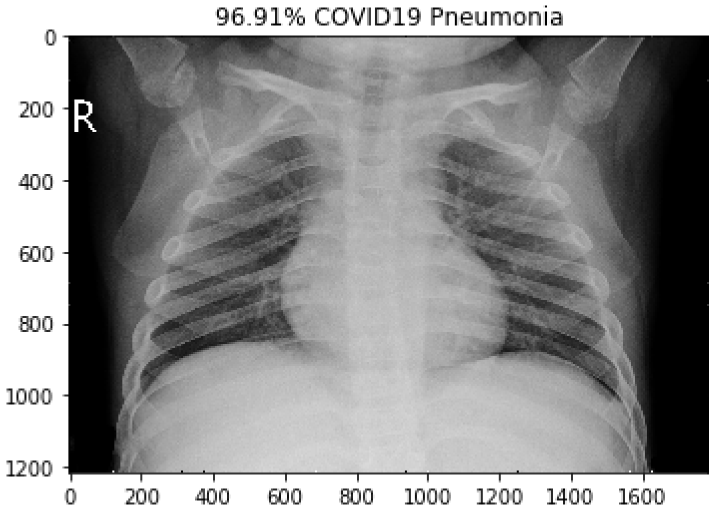
normal₩NORMAL2-IM-1401-0001.jpeg



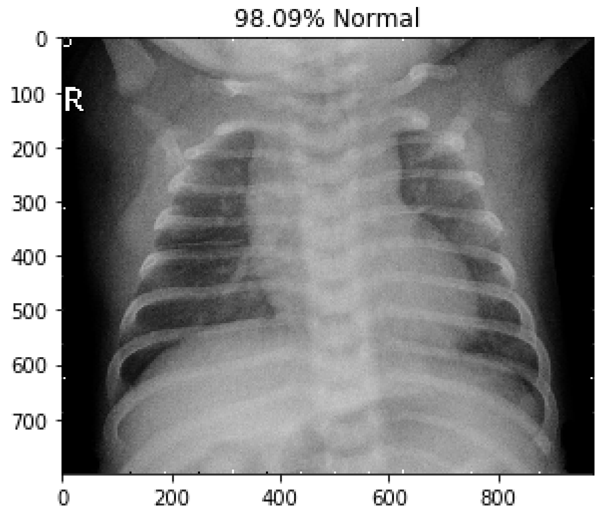
normal₩NORMAL2-IM-1406-0001.jpeg



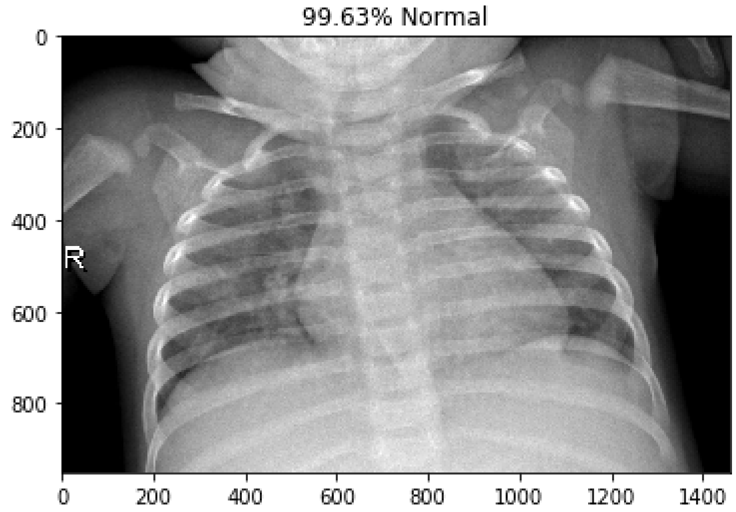
normal₩NORMAL2-IM-1412-0001.jpeg



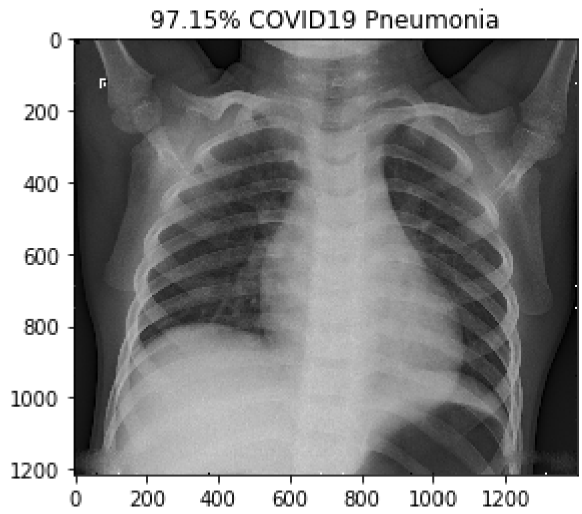

```
normal\NORMAL2-IM-1419-0001.jpeg
```




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




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



[10 points] TSNE Plot

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

intermediate_layer_model = models.Model(inputs=model.input,
                                        outputs=model.get_layer('dense_feature').output)
tsne_data_generator = test_datagen.flow_from_directory(DATASET_PATH,target_size=IMAGE_SIZE,
                                                       batch_size=1,shuffle=True,seed=42,class_mode=
"binary")


labels = []
num = len(tsne_data_generator)
for i in range(num):
    labels.extend(np.array(tsne_data_generator[i][1]))

#feature extraction
tsne_data_generator.reset()
features = intermediate_layer_model.predict_generator(tsne_data_generator)

#compress the dimensionality
tsne = TSNE(n_components=2)
tsne_result = tsne.fit_transform(features)
x=[i[0] for i in tsne_result]
y=[i[1] for i in tsne_result]

#plotting values
covid_x = []
covid_y = []
non_covid_x = []
non_covid_y = []
for i in range(len(labels)):
    if labels[i] == 0.0:
        non_covid_x.append(x[i])
        non_covid_y.append(y[i])
    else:
        covid_x.append(x[i])
        covid_y.append(y[i])

#plotting
plt.scatter(non_covid_x,non_covid_y, label='Normal')
plt.scatter(covid_x,covid_y, label='COVID-19')
plt.legend(loc='upper left')
plt.show()
```

```
Found 130 images belonging to 2 classes.
130
```