Ziqi_Tan_task1

April 22, 2020

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

2 Task 1 [Total points: 30]

2.1 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
```

2.2 Data

Please download the data using the following link: COVID-19.

- 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.

2.3 [20 points] Binary Classification: COVID-19 vs. Normal

[1]:
```
import os

import tensorflow as tf
```

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

from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

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

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 12801178641409884498
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 4930941747
locality {
  bus_id: 1
  links {
  }
}
incarnation: 16780514319106285150
physical_device_desc: "device: 0, name: GeForce GTX 1060, pci bus id:
0000:01:00.0, compute capability: 6.1"
]
```

[1]: '2.1.0'

Load Image Data

[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 memory
NUM_EPOCHS    = 40
LEARNING_RATE = 0.0005 # start off with high rate first 0.001 and experiment␣
 ↪with reducing it gradually
```

Generate Training and Validation Batches

[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")
```

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

```
C:\Users\tanzi\Anaconda3\lib\site-
packages\keras_preprocessing\image\image_data_generator.py:341: UserWarning:
This ImageDataGenerator specifies `zca_whitening` which overrides setting
of`featurewise_std_normalization`.
  warnings.warn('This ImageDataGenerator specifies '
```

[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.

[4]:
```
# raise NotImplementedError("Build your model based on an architecture of your␣
↪choice "
#                           "A sample model summary is shown below")

# Implement VGG16
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Sequential
```

```python
# vgg_16 = VGG16(include_top=False, weights='imagenet', input_tensor=None,␣
↪input_shape=None, pooling=None, classes=1000)
vgg_16 = VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3),␣
↪pooling='None', classes=2)
# Arguments
    # include_top: whether to include the 3 fully-connected layers at the top of␣
↪the network.
    # weights: one of None (random initialization) or 'imagenet' (pre-training␣
↪on ImageNet).
    # input_tensor: optional Keras tensor (i.e. output of layers.Input()) to use␣
↪as image input for the model.
    # input_shape: optional shape tuple,
                    # only to be specified if include_top is False
                    # (otherwise the input shape has to be (224, 224, 3) (with␣
↪'channels_last' data format)
                    # or (3, 224, 224) (with 'channels_first' data format).
                    # It should have exactly 3 inputs channels,
                    # and width and height should be no smaller than 32. E.g.␣
↪(200, 200, 3) would be one valid value.
    # pooling: Optional pooling mode for feature extraction when include_top is␣
↪False.
                    # None means that the output of the model will be the 4D␣
↪tensor output of the last convolutional block.
                    # 'avg' means that global average pooling will be applied to␣
↪the output of the last convolutional block,
                    # and thus the output of the model will be a 2D tensor.
                    # 'max' means that global max pooling will be applied.
    # classes: optional number of classes to classify images into,
                    # only to be specified if include_top is True,
                    # and if no weights argument is specified.

vgg_16.trainable = False

covid_model = Sequential()
covid_model.add(vgg_16)
covid_model.add(Flatten())
covid_model.add(Dense(256, activation='relu'))
covid_model.add(Dense(1, activation='sigmoid'))

covid_model.build(input_shape=(224, 224, 3))
covid_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
```

```
_________________________________________________________________
flatten (Flatten)            (None, 25088)            0
_________________________________________________________________
dense (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

[5]:
```python
# 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

# raise NotImplementedError("Use the model.fit function to train your network")
covid_model.compile(optimizer='adam', loss=tf.keras.losses.
 ↪BinaryCrossentropy(from_logits=False), metrics=['accuracy'])

# print the device library
print(device_lib.list_local_devices())

history = None

with tf.device("GPU:0"):
    history = covid_model.fit_generator(generator=train_batches,
                                        steps_per_epoch=STEP_SIZE_TRAIN,
                                        epochs=40,
                                        validation_data=(valid_batches),
                                        validation_steps=STEP_SIZE_VALID)
```

```
11
3
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 4364867676804776945
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 4930941747
```

```
locality {
  bus_id: 1
  links {
  }
}
incarnation: 1115653898984476423
physical_device_desc: "device: 0, name: GeForce GTX 1060, pci bus id:
0000:01:00.0, compute capability: 6.1"
]
WARNING:tensorflow:From <ipython-input-5-ba369a66ef5d>:21: Model.fit_generator
(from tensorflow.python.keras.engine.training) is deprecated and will be removed
in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
```

```
C:\Users\tanzi\Anaconda3\lib\site-
packages\keras_preprocessing\image\image_data_generator.py:716: UserWarning:
This ImageDataGenerator specifies `featurewise_center`, but it hasn't been fit
on any training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
C:\Users\tanzi\Anaconda3\lib\site-
packages\keras_preprocessing\image\image_data_generator.py:735: UserWarning:
This ImageDataGenerator specifies `zca_whitening`, but it hasn't been fit on any
training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 10 steps, validate for 2 steps
Epoch 1/40
```

```
C:\Users\tanzi\Anaconda3\lib\site-
packages\keras_preprocessing\image\image_data_generator.py:716: UserWarning:
This ImageDataGenerator specifies `featurewise_center`, but it hasn't been fit
on any training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
C:\Users\tanzi\Anaconda3\lib\site-
packages\keras_preprocessing\image\image_data_generator.py:735: UserWarning:
This ImageDataGenerator specifies `zca_whitening`, but it hasn't been fit on any
training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
10/10 [==============================] - 5s 526ms/step - loss: 2.1802 -
accuracy: 0.5638 - val_loss: 0.1493 - val_accuracy: 1.0000
```

```
Epoch 2/40
10/10 [==============================] - 4s 379ms/step - loss: 0.7641 -
accuracy: 0.7660 - val_loss: 0.2696 - val_accuracy: 0.9000
Epoch 3/40
10/10 [==============================] - 4s 363ms/step - loss: 0.4144 -
accuracy: 0.8191 - val_loss: 0.1002 - val_accuracy: 0.9500
Epoch 4/40
10/10 [==============================] - 4s 376ms/step - loss: 0.1796 -
accuracy: 0.9149 - val_loss: 0.1578 - val_accuracy: 0.9500
Epoch 5/40
10/10 [==============================] - 4s 389ms/step - loss: 0.1891 -
accuracy: 0.9255 - val_loss: 0.2265 - val_accuracy: 0.9500
Epoch 6/40
10/10 [==============================] - 4s 367ms/step - loss: 0.1898 -
accuracy: 0.9149 - val_loss: 0.1955 - val_accuracy: 0.9000
Epoch 7/40
10/10 [==============================] - 4s 397ms/step - loss: 0.1080 -
accuracy: 0.9500 - val_loss: 0.2077 - val_accuracy: 0.9000
Epoch 8/40
10/10 [==============================] - 4s 379ms/step - loss: 0.2766 -
accuracy: 0.9149 - val_loss: 0.0570 - val_accuracy: 0.9500
Epoch 9/40
10/10 [==============================] - 4s 368ms/step - loss: 0.3627 -
accuracy: 0.8617 - val_loss: 0.3945 - val_accuracy: 0.9500
Epoch 10/40
10/10 [==============================] - 4s 372ms/step - loss: 0.1822 -
accuracy: 0.9255 - val_loss: 0.1689 - val_accuracy: 0.9000
Epoch 11/40
10/10 [==============================] - 4s 397ms/step - loss: 0.1349 -
accuracy: 0.9500 - val_loss: 0.0183 - val_accuracy: 1.0000
Epoch 12/40
10/10 [==============================] - 4s 384ms/step - loss: 0.1786 -
accuracy: 0.9043 - val_loss: 0.0782 - val_accuracy: 0.9500
Epoch 13/40
10/10 [==============================] - 4s 370ms/step - loss: 0.3579 -
accuracy: 0.9043 - val_loss: 0.0164 - val_accuracy: 1.0000
Epoch 14/40
10/10 [==============================] - 4s 399ms/step - loss: 0.1380 -
accuracy: 0.9400 - val_loss: 0.2965 - val_accuracy: 0.8000
Epoch 15/40
10/10 [==============================] - 4s 378ms/step - loss: 0.1631 -
accuracy: 0.9574 - val_loss: 0.3378 - val_accuracy: 0.9000
Epoch 16/40
10/10 [==============================] - 4s 372ms/step - loss: 0.1677 -
accuracy: 0.9468 - val_loss: 0.0537 - val_accuracy: 0.9500
Epoch 17/40
10/10 [==============================] - 4s 375ms/step - loss: 0.1484 -
accuracy: 0.9255 - val_loss: 0.1256 - val_accuracy: 0.9500
```

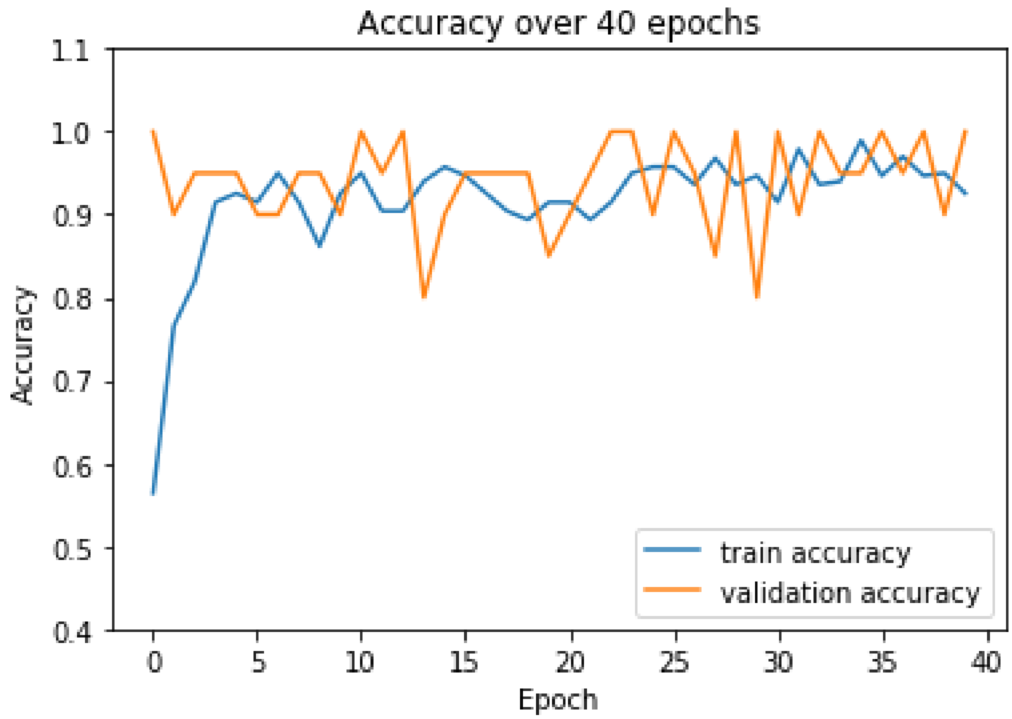
```
Epoch 18/40
10/10 [==============================] - 4s 449ms/step - loss: 0.3752 -
accuracy: 0.9043 - val_loss: 0.0712 - val_accuracy: 0.9500
Epoch 19/40
10/10 [==============================] - 4s 367ms/step - loss: 0.2058 -
accuracy: 0.8936 - val_loss: 0.0876 - val_accuracy: 0.9500
Epoch 20/40
10/10 [==============================] - 4s 386ms/step - loss: 0.2932 -
accuracy: 0.9149 - val_loss: 0.3715 - val_accuracy: 0.8500
Epoch 21/40
10/10 [==============================] - 4s 396ms/step - loss: 0.2374 -
accuracy: 0.9149 - val_loss: 0.3073 - val_accuracy: 0.9000
Epoch 22/40
10/10 [==============================] - 4s 392ms/step - loss: 0.3476 -
accuracy: 0.8936 - val_loss: 0.2592 - val_accuracy: 0.9500
Epoch 23/40
10/10 [==============================] - 4s 418ms/step - loss: 0.1627 -
accuracy: 0.9149 - val_loss: 0.0315 - val_accuracy: 1.0000
Epoch 24/40
10/10 [==============================] - 4s 402ms/step - loss: 0.1275 -
accuracy: 0.9500 - val_loss: 0.0098 - val_accuracy: 1.0000
Epoch 25/40
10/10 [==============================] - 4s 397ms/step - loss: 0.1210 -
accuracy: 0.9574 - val_loss: 0.2710 - val_accuracy: 0.9000
Epoch 26/40
10/10 [==============================] - 4s 385ms/step - loss: 0.0743 -
accuracy: 0.9574 - val_loss: 0.0625 - val_accuracy: 1.0000
Epoch 27/40
10/10 [==============================] - 4s 376ms/step - loss: 0.1272 -
accuracy: 0.9362 - val_loss: 0.1396 - val_accuracy: 0.9500
Epoch 28/40
10/10 [==============================] - 4s 377ms/step - loss: 0.1329 -
accuracy: 0.9681 - val_loss: 0.2069 - val_accuracy: 0.8500
Epoch 29/40
10/10 [==============================] - 4s 417ms/step - loss: 0.1452 -
accuracy: 0.9362 - val_loss: 0.0255 - val_accuracy: 1.0000
Epoch 30/40
10/10 [==============================] - 4s 386ms/step - loss: 0.2279 -
accuracy: 0.9468 - val_loss: 0.7686 - val_accuracy: 0.8000
Epoch 31/40
10/10 [==============================] - 4s 388ms/step - loss: 0.2312 -
accuracy: 0.9149 - val_loss: 0.0241 - val_accuracy: 1.0000
Epoch 32/40
10/10 [==============================] - 4s 376ms/step - loss: 0.0722 -
accuracy: 0.9787 - val_loss: 0.1561 - val_accuracy: 0.9000
Epoch 33/40
10/10 [==============================] - 4s 381ms/step - loss: 0.1511 -
accuracy: 0.9362 - val_loss: 0.0364 - val_accuracy: 1.0000
```

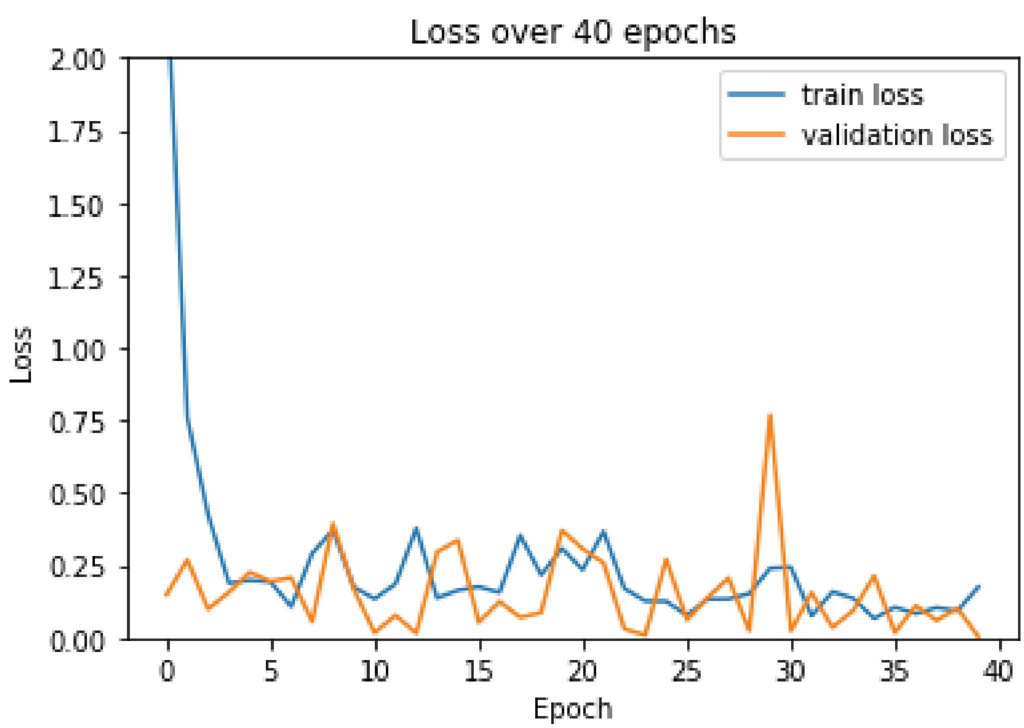
```
Epoch 34/40
10/10 [==============================] - 4s 398ms/step - loss: 0.1366 -
accuracy: 0.9400 - val_loss: 0.0953 - val_accuracy: 0.9500
Epoch 35/40
10/10 [==============================] - 4s 371ms/step - loss: 0.0637 -
accuracy: 0.9894 - val_loss: 0.2142 - val_accuracy: 0.9500
Epoch 36/40
10/10 [==============================] - 4s 396ms/step - loss: 0.1004 -
accuracy: 0.9468 - val_loss: 0.0184 - val_accuracy: 1.0000
Epoch 37/40
10/10 [==============================] - 4s 401ms/step - loss: 0.0842 -
accuracy: 0.9700 - val_loss: 0.1114 - val_accuracy: 0.9500
Epoch 38/40
10/10 [==============================] - 4s 393ms/step - loss: 0.0997 -
accuracy: 0.9468 - val_loss: 0.0607 - val_accuracy: 1.0000
Epoch 39/40
10/10 [==============================] - 4s 419ms/step - loss: 0.0937 -
accuracy: 0.9500 - val_loss: 0.1029 - val_accuracy: 0.9000
Epoch 40/40
10/10 [==============================] - 4s 387ms/step - loss: 0.1678 -
accuracy: 0.9255 - val_loss: 0.0046 - val_accuracy: 1.0000
```

[5 points] Plot Accuracy and Loss During Training

[6]:
```python
import matplotlib.pyplot as plt

# raise NotImplementedError("Plot the accuracy and the loss during training")

# Accuracy over 40 Epochs
plt.figure()
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label = 'validation accuracy')
plt.title('Accuracy over 40 epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.4, 1.1])
plt.legend(loc='lower right')

# Loss over 40 Epochs
plt.figure()
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label = 'validation loss')
plt.title('Loss over 40 epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 2.0])
plt.legend(loc='upper right')
```

```
[6]: <matplotlib.legend.Legend at 0x2b46b82bb70>
```

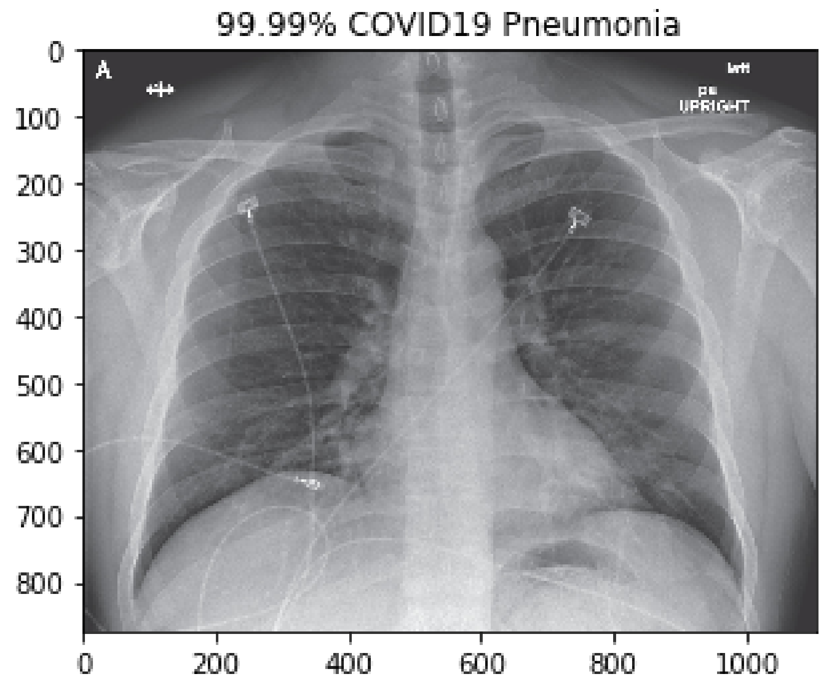




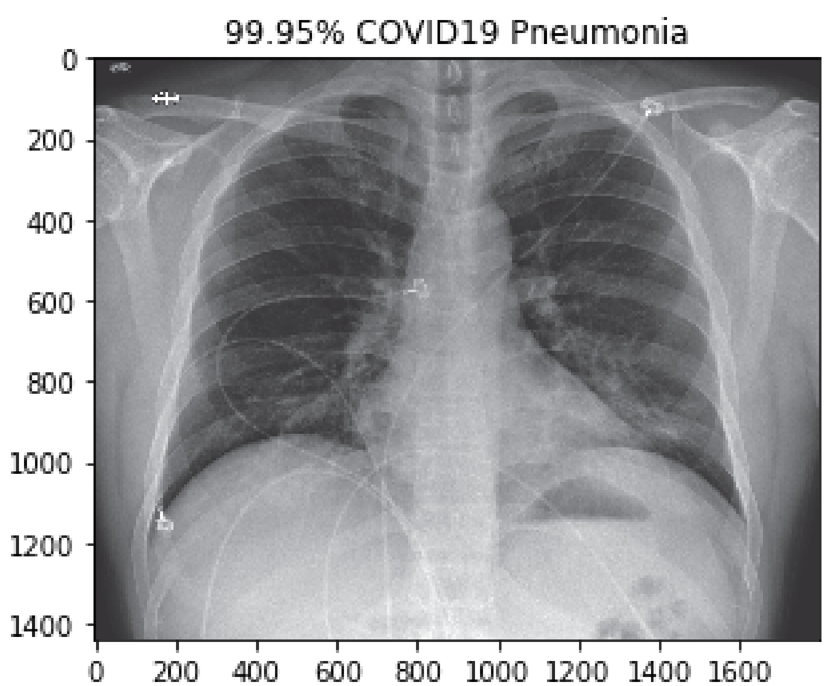
Plot Test Results

[7]:
```
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)
pred = covid_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()
```

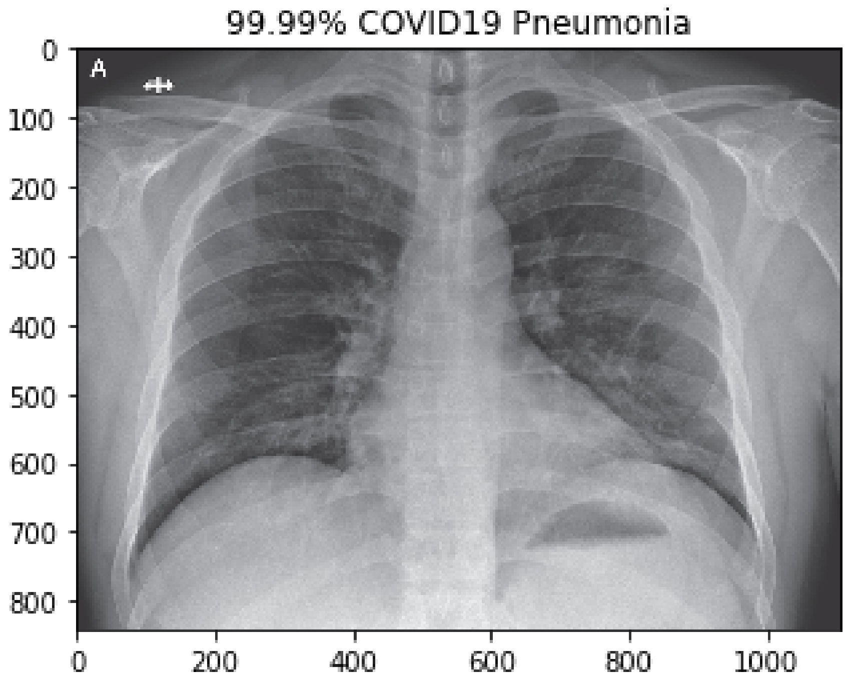
```
Found 18 images belonging to 2 classes.
WARNING:tensorflow:From <ipython-input-7-e21232c472f7>:9:
Model.predict_generator (from tensorflow.python.keras.engine.training) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
18/18 [==============================] - 1s 50ms/step
covid\nejmoa2001191_f3-PA.jpeg
```



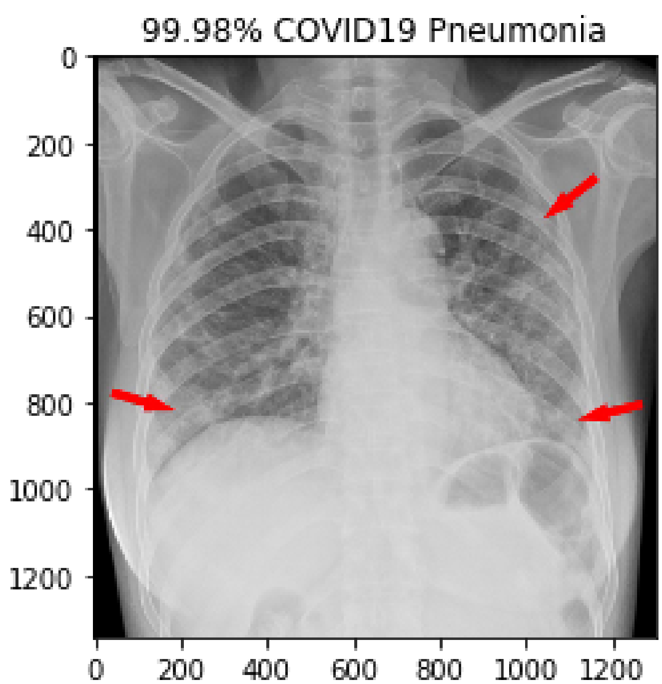
covid\nejmoa2001191_f4.jpeg



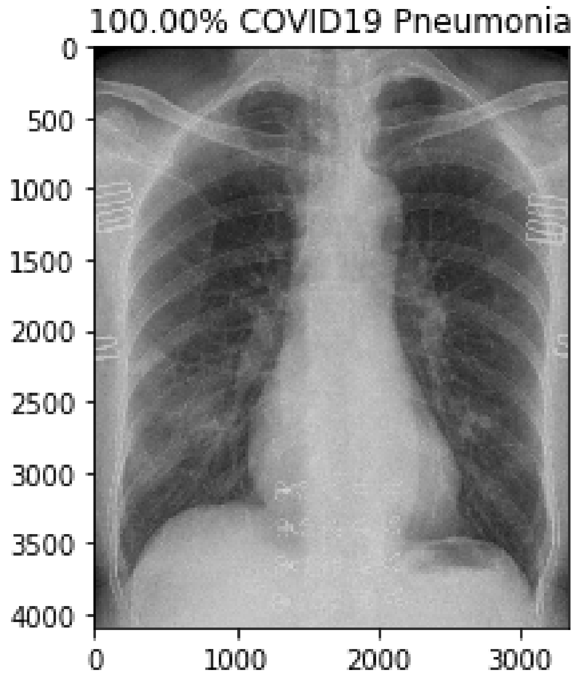
covid\nejmoa2001191_f5-PA.jpeg



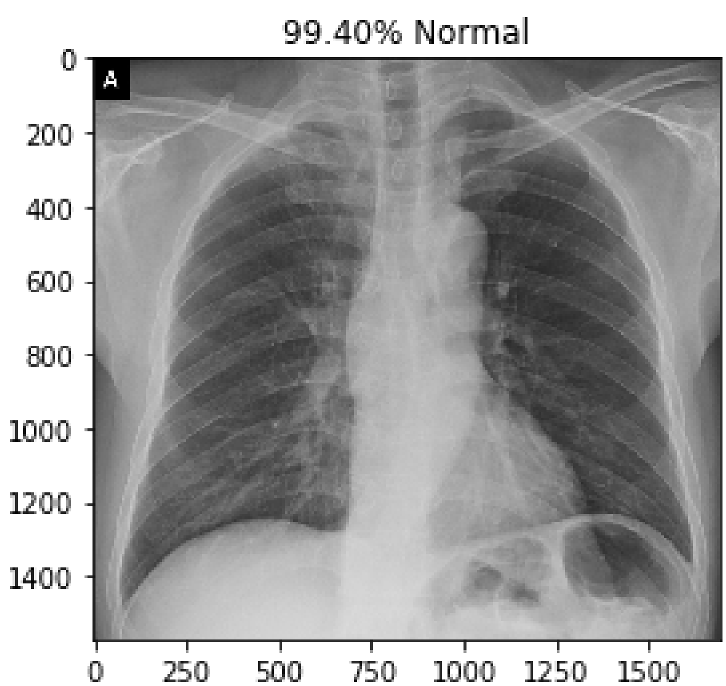
covid\radiol.2020200490.fig3.jpeg



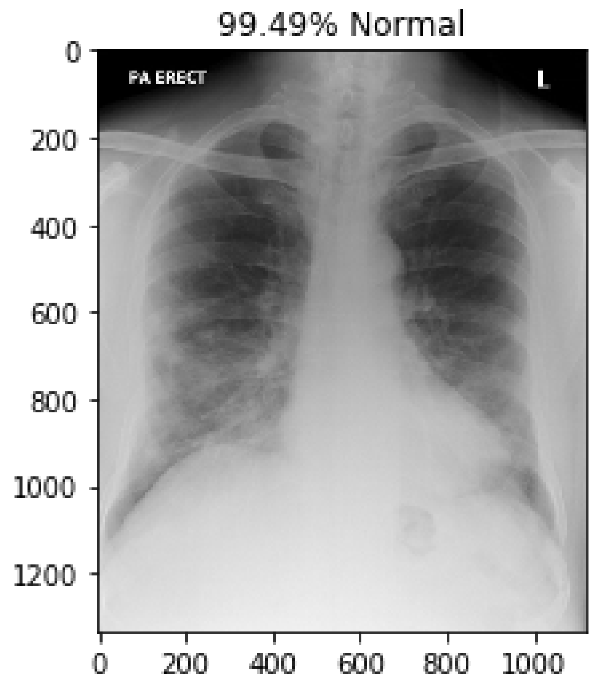
covid\ryct.2020200028.fig1a.jpeg



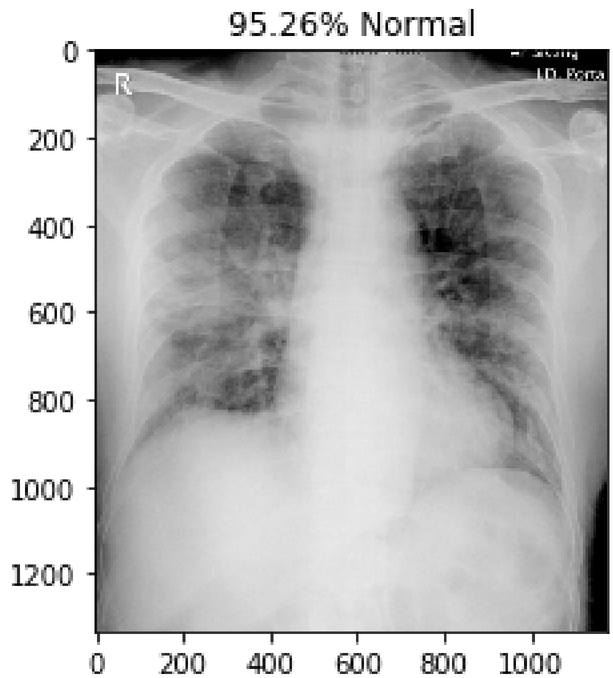
covid\ryct.2020200034.fig2.jpeg



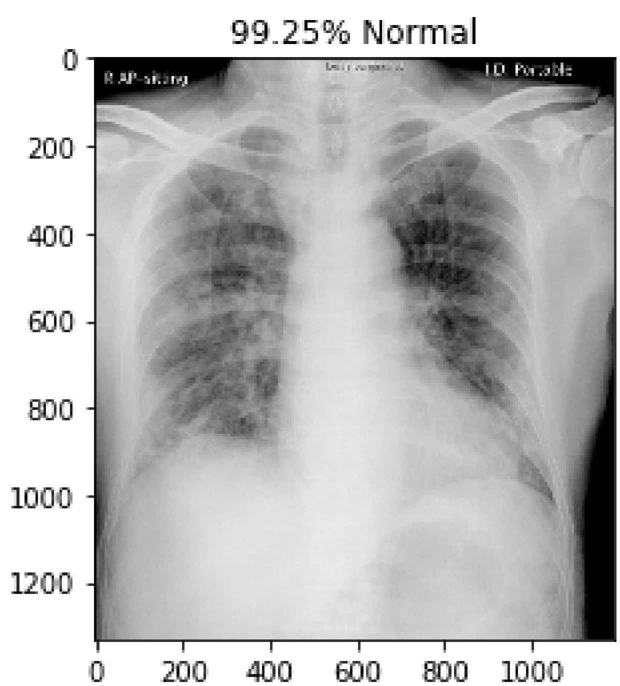
covid\ryct.2020200034.fig5-day0.jpeg



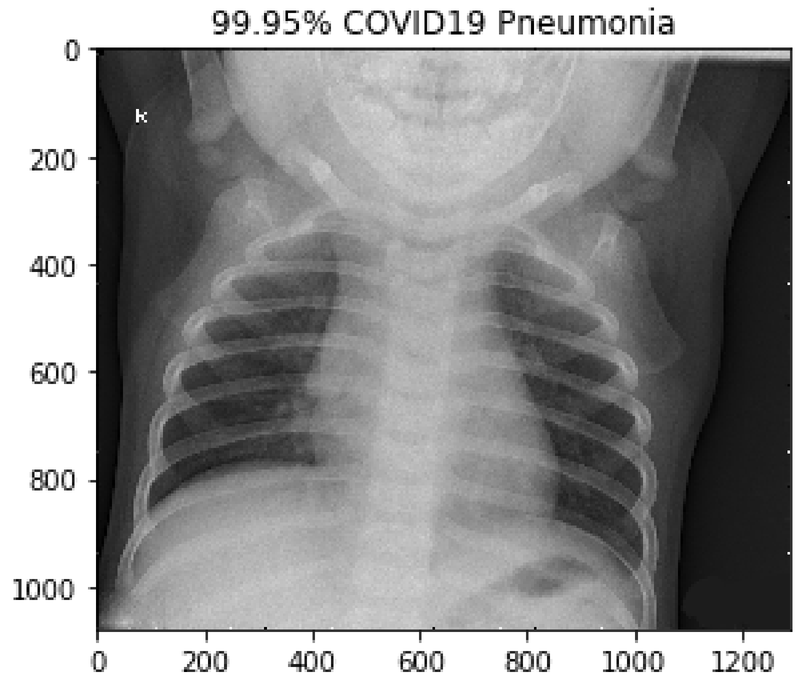
covid\ryct.2020200034.fig5-day4.jpeg



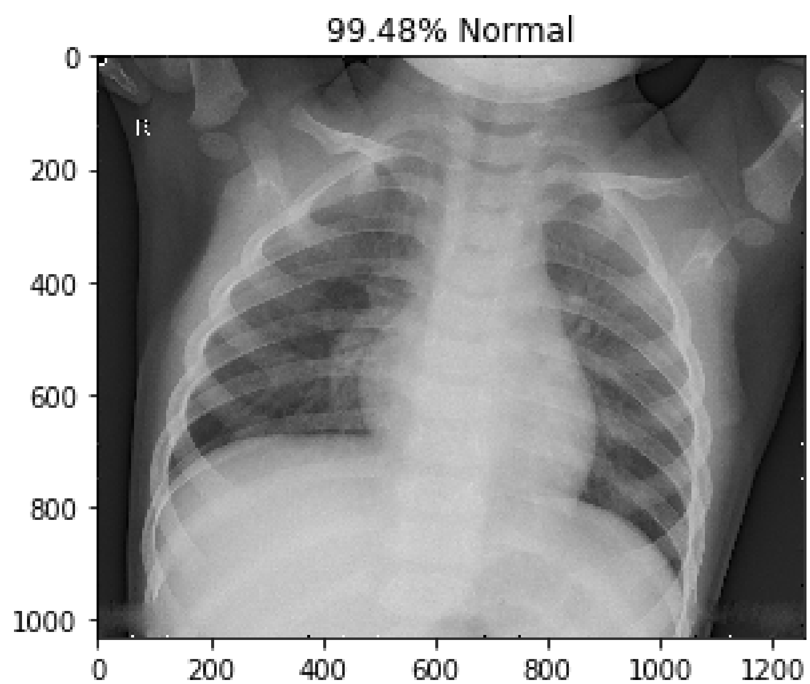
covid\ryct.2020200034.fig5-day7.jpeg



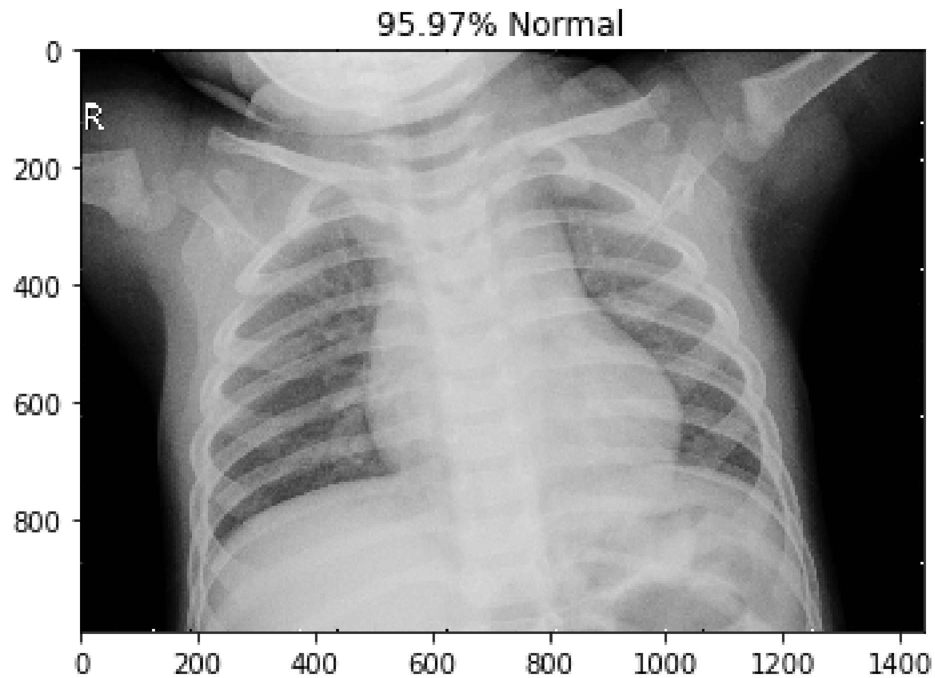
normal\NORMAL2-IM-1385-0001.jpeg



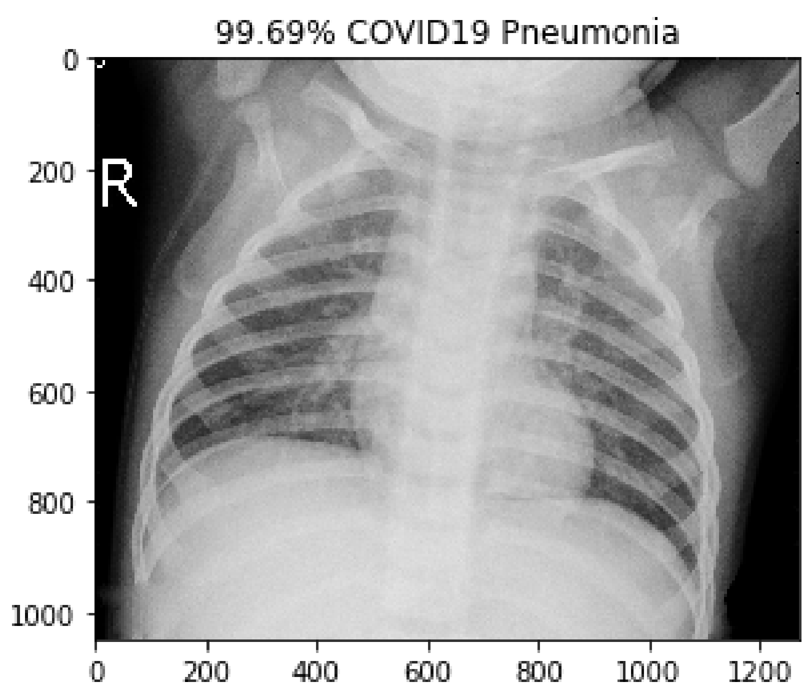
normal\NORMAL2-IM-1396-0001.jpeg



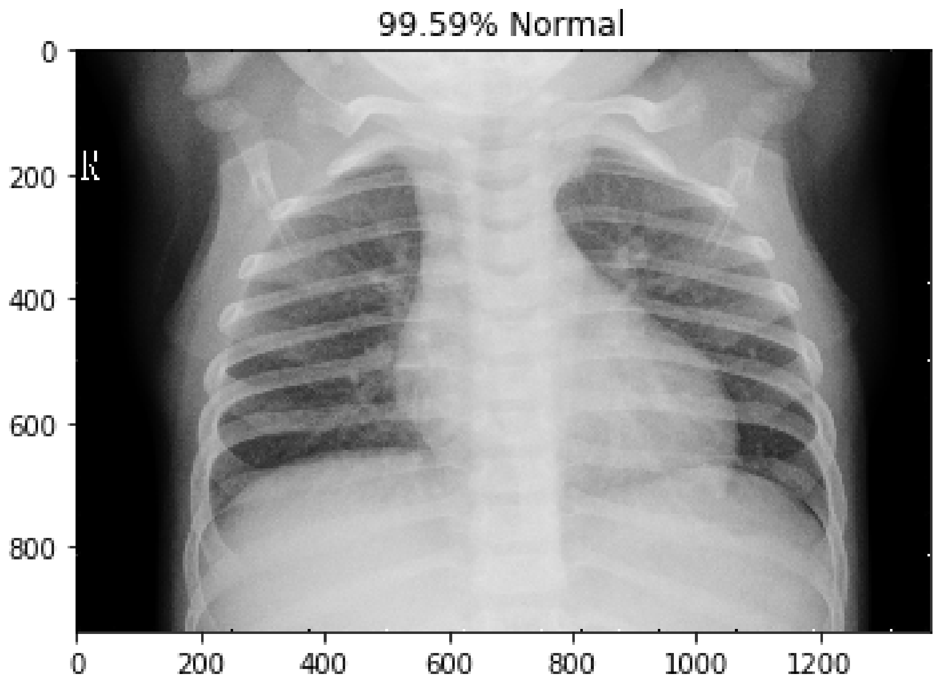
normal\NORMAL2-IM-1400-0001.jpeg



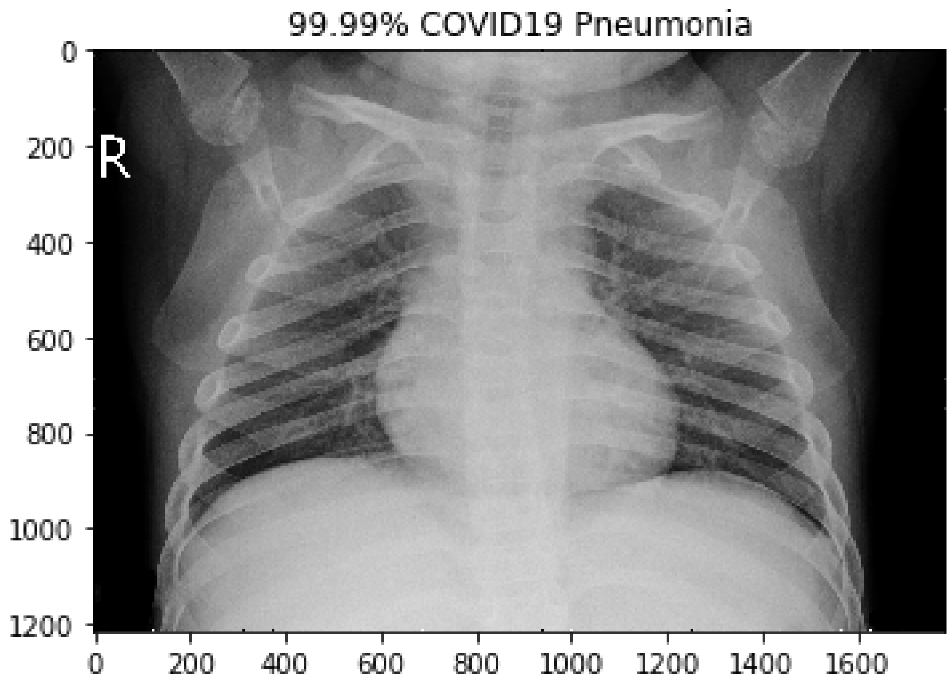
normal\NORMAL2-IM-1401-0001.jpeg



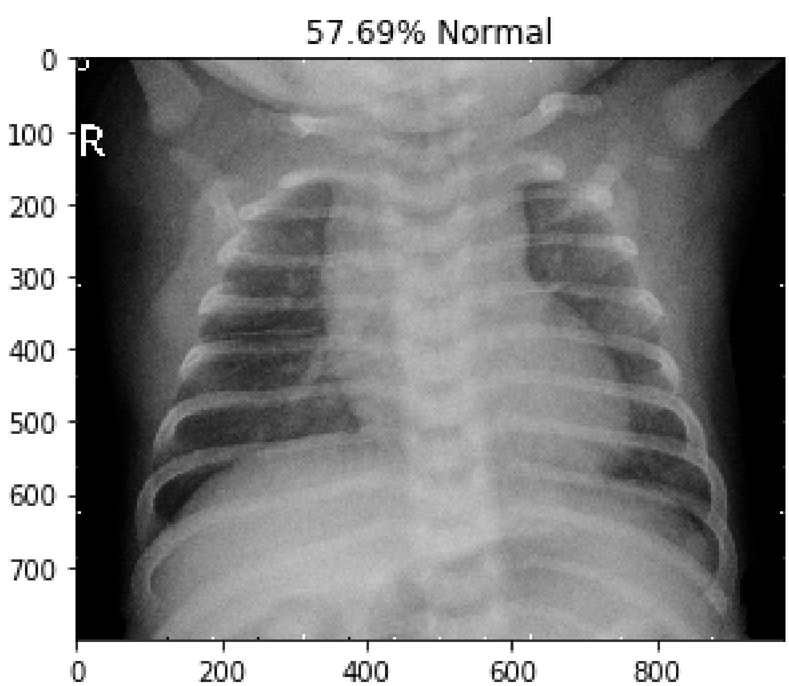
normal\NORMAL2-IM-1406-0001.jpeg



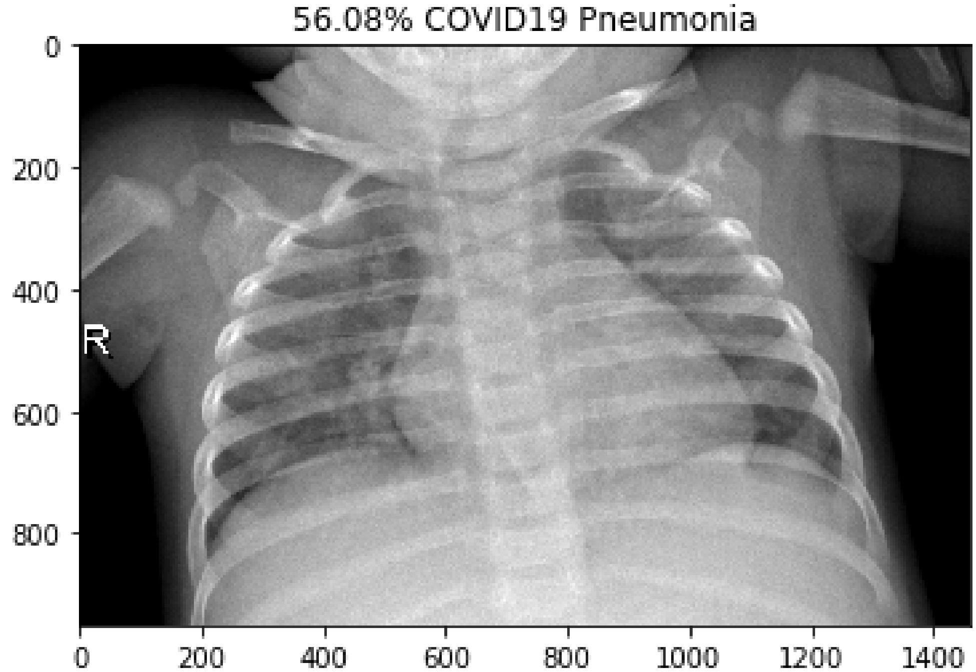
normal\NORMAL2-IM-1412-0001.jpeg



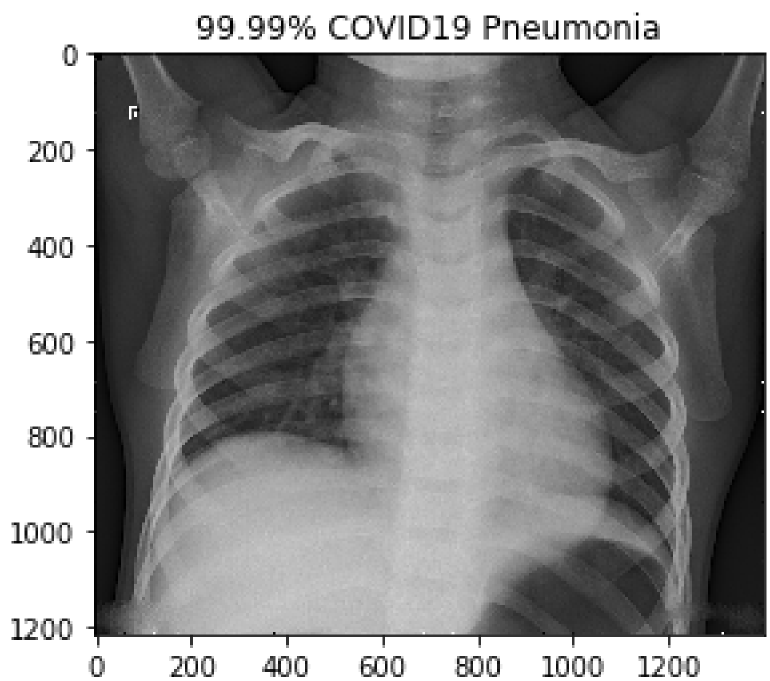
normal\NORMAL2-IM-1419-0001.jpeg



normal\NORMAL2-IM-1422-0001.jpeg



normal\NORMAL2-IM-1423-0001.jpeg



2.4 [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.

[9]:
```
from sklearn.manifold import TSNE

intermediate_layer_model = tf.keras.models.Model(inputs=covid_model.input,
                                                 outputs=covid_model.get_layer('dense').
 ↪output)
tsne_data_generator = test_datagen.
 ↪flow_from_directory(DATASET_PATH,target_size=IMAGE_SIZE,
                                                            ␣
 ↪batch_size=1,shuffle=False,seed=42,class_mode="binary")

# raise NotImplementedError("Extract features from the tsne_data_generator and␣
 ↪fit a t-SNE model for the features,"
#                            "and plot the resulting 2D features of the two␣
 ↪classes.")

outputs = intermediate_layer_model.
 ↪predict_generator(tsne_data_generator,130,verbose=1)
print(outputs.shape)
label = tsne_data_generator.classes
features = TSNE(n_components=2).fit_transform(outputs)
print(features.shape)

covid_x = []
covid_y = []
normal_x = []
normal_y = []

plt.figure()
for index in range(len(features)):
    if label[index] == 0:
        # COVID
        covid_x.append(features[index, 0])
        covid_y.append(features[index, 1])
    else:
        # normal
        normal_x.append(features[index, 0])
```

```
        normal_y.append(features[index, 1])


plt.title('2D features')
plt.plot(covid_x, covid_y, 'ro', label="COVID-19")
plt.plot(normal_x, normal_y, 'bo', label="Normal")
plt.legend(loc='upper right')
```

```
Found 130 images belonging to 2 classes.
130/130 [==============================] - 4s 27ms/step
(130, 256)
(130, 2)
```

[9]: <matplotlib.legend.Legend at 0x2b4930c62e8>