Detecting COVID-19 induced Pneumonia from Chest X-rays with Transfer Learning: An implementation in Tensorflow and Keras.

Deep Learning on the march against the novel coronavirus.

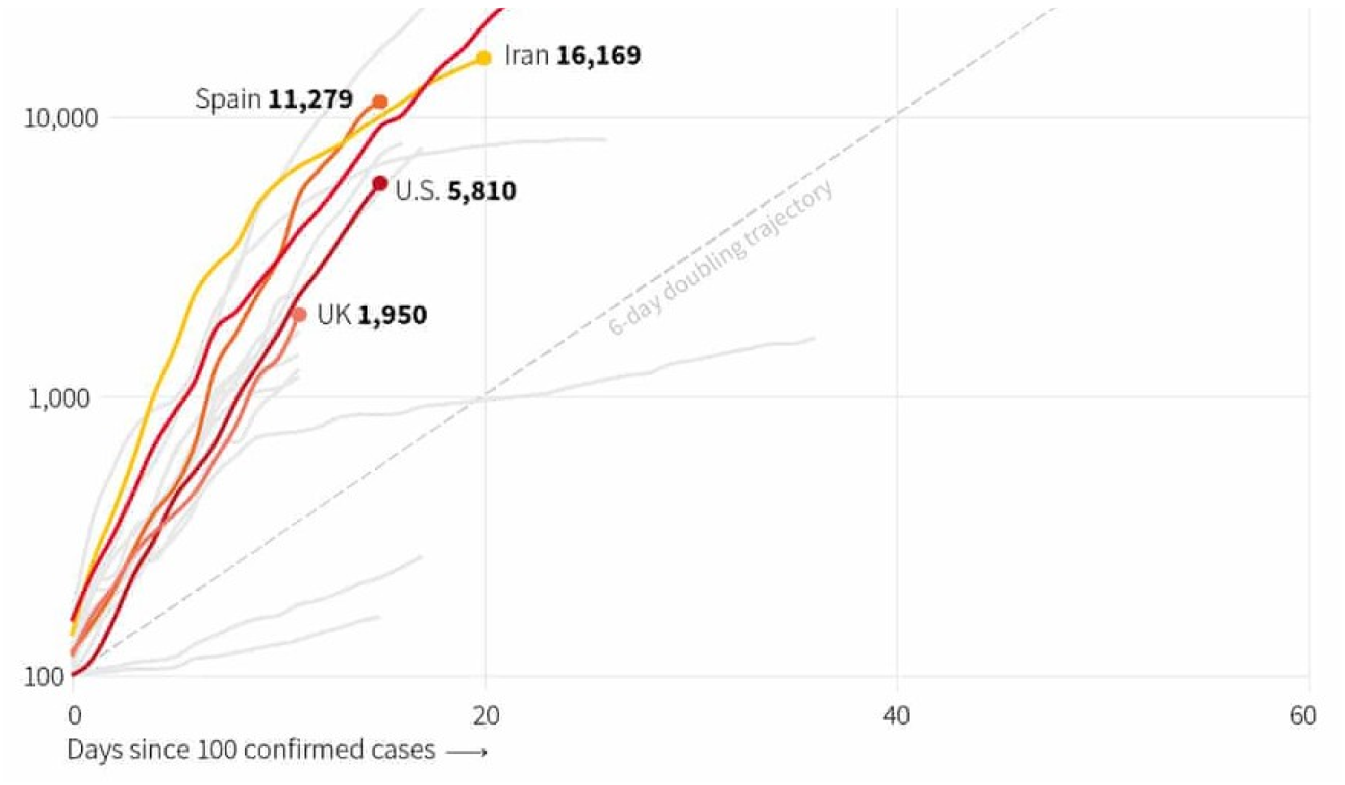
Adrian Yijie Xu Mar 21, 2020 · 8 min read ★

Introduction

The novel coronavirus (SARS-COV-2) has become the most pressing issue facing mankind. Like a wildfire burning through the world, the COVID-19 disease has changed the global landscape in only three months. A high reproduction rate and a higher chance of complications has led to border closures, empty streets, rampant stockpiling, mass self-isolation policies, an economic recession, and a cultural shift towards mistrust. **The world has entered into a new World War, Blitzkrieg-ed by an invisible enemy.**

With the number of cases accelerating across the developed world, governments are slowly awakening to the scale of the issue, and one would hope to see the world united against such a threat. However, at both a national and individual-level, we're still observing divisions and mistrust. Reports of racist attacks, theft, and irresponsible behaviour are rampant. Doctor's are pleading for the public to take the situation seriously, and to shelter-in-place while avoiding all non-essential travel.

100,000 cases of COVID-19
Mainland China
80,860 cases
Italy 31,506



Number of cases of COVID19 for a selected number of nations. (Source: Reuters)

Strategies to combat the virus prior to March have primarily revolved around containment and tracing — to find and isolate possible cases before they develop. However, with the exponential increase in cases, hospitals around the world have rapidly become overwhelmed, and government policies have shifted towards mitigation or even acceptance. The frustrating process is further compounded by a **lack of testing capacity in the affected nations**, resulting in the number of confirmed cases lagging behind the true value. This led to the health authorities in China to temporarily revert to diagnosing based on symptoms and Chest X-rays/CTs. However, such solutions are still at risk of overlapping with other causes of pneumonia, they led to much confusion, and the nation reverted back to PCR (polymerase chain reaction) — based testing.

AI-based solutions have recently been extensively explored for the use of pneumonia detection in studies, and such approaches have also been proposed during this pandemic. Custom built models have been reported of achieving sensitivity and specificity rates of approaching 90%. However, their specificity in identifying the cause of pneumonia, particularly against other viral illnesses (such as influenza), are still being hotly debated. Their inherent experimental nature requires that confirmatory approaches are also used in tandem, as part of a multidisciplinary approach for diagnosis.

Transfer Learning process and a constrained dataset. Naturally, the methods and described here are meant for educational purposes only. This is not a scientifically rigorous study, and should not be used for medical purposes.

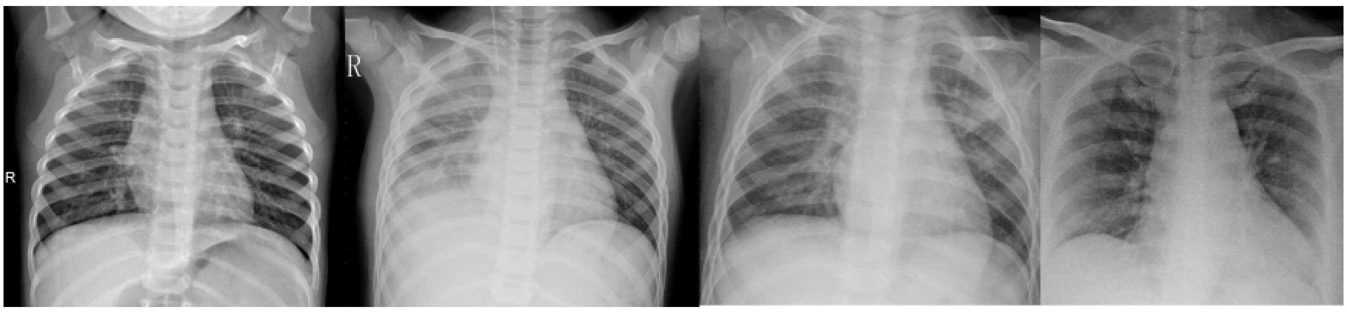
Implementation

We've covered Transfer Learning and Convolutional Neural Networks in detail in previous articles, and highly recommend our readers to consult them for a more thorough consideration.

To target the issue at hand, we've collected own dataset,combining the Kaggle Chest X-ray dataset with the COVID19 Chest X-ray dataset collected by Dr. Joseph Paul Cohen of the University of Montreal. Both of these datasets consist of posterior anterior chest images of patients with pneumonia. As the COVID19 dataset is being updated daily as more cases are published, we accessed the instance available on the 18th of March, 2020. Our dataset is split into 4 different categories, with 9 images per class used as a test set.

- Healthy: 79 images
- Pneumonia (Viral) : 79 images
- Pneumonia (Bacterial): 79 images
- Pneumonia (COVID-19): 69 images

Let's take a a look at some examples to emphasize the minute differences between the different causes. In particular, **the differences between the Viral and COVID-19 cases are indistinguishable without extensive radiological training, reinforcing the difficulties faced by healthcare providers on the front line**



PA Chest X-rays of admitted patients. From Left to Right: Healthy, Bacterial, Viral, COVID19.

The Binary Case

Let’s begin with the binary case — comparing healthy lungs to those exhibiting pneumonia caused by the SARS-COV-2 virus. This is covered in the *Covid19_GradientCrescent_Binary* notebook.

To begin with, let’s import our dataset from the GradientCrescent Google Drive.

```
!gdown https://drive.google.com/uc?id=1coM7x3378f-
Ou2l6Pg2wldaOI7Dntu1a

!unzip Covid_Data_GradientCrescent.zip
```

Next, we’ll import some of the necessary libraries, and define our dataset paths and a few parameters of our network. As we are transfer learning, we’ll keep our learning rate at a low value of 5e-4.

```
import numpy as np

import tensorflow as tf

DATASET_PATH = ‘/content/two/train’

test_dir = ‘/content/two/test’

IMAGE_SIZE = (150, 150)

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 = 20

LEARNING_RATE =0.0005
```

Next, let’s setup the respective training and validation preprocessing and batch image preparation functions using the ImageDataGenerator class, specifying our *class_mode* parameter as “binary” for this case.

```
 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",


 )
```

With all of this complete, let's define our network, which is shared between both of our notebooks. We're taking a pre-trained VGG16 network, and fitting it to a series of densely-connected layers of our own, with our output layer featuring a sigmoid activation function for the binary classification case. We'll compile our network and attach the ADAM optimizer to it.

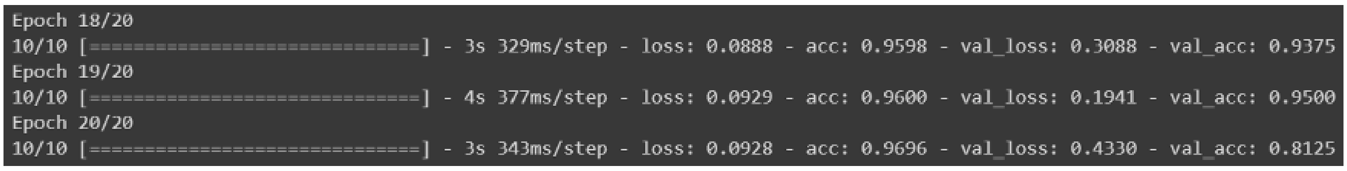
```
from keras import models
from keras import layers
from keras.applications import VGG16
from keras import optimizers
from keras.layers.core import Flatten, Dense, Dropout, Lambda
```

```
conv_base.trainable = False

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
 optimizer=optimizers.Adam(lr=LEARNING_RATE),
 metrics=['acc'])
```

With our network defined, let's begin training for 20 epochs.

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

result=model.fit_generator(train_batches,
 steps_per_epoch =STEP_SIZE_TRAIN,
 validation_data = valid_batches,
 validation_steps = STEP_SIZE_VALID,
 epochs= NUM_EPOCHS,
 )
```

You should find that our validation accuracy converges relatively quickly to beyond 80%. With a larger dataset, we'd be able to feed more validation examples to reduce the inter-epoch variation, but the results are satisfactory for demonstration purposes.



```
Epoch 18/20
10/10 [==============================] - 3s 329ms/step - loss: 0.0888 - acc: 0.9598 - val_loss: 0.3088 - val_acc: 0.9375
Epoch 19/20
10/10 [==============================] - 4s 377ms/step - loss: 0.0929 - acc: 0.9600 - val_loss: 0.1941 - val_acc: 0.9500
Epoch 20/20
10/10 [==============================] - 3s 343ms/step - loss: 0.0928 - acc: 0.9696 - val_loss: 0.4330 - val_acc: 0.8125
```

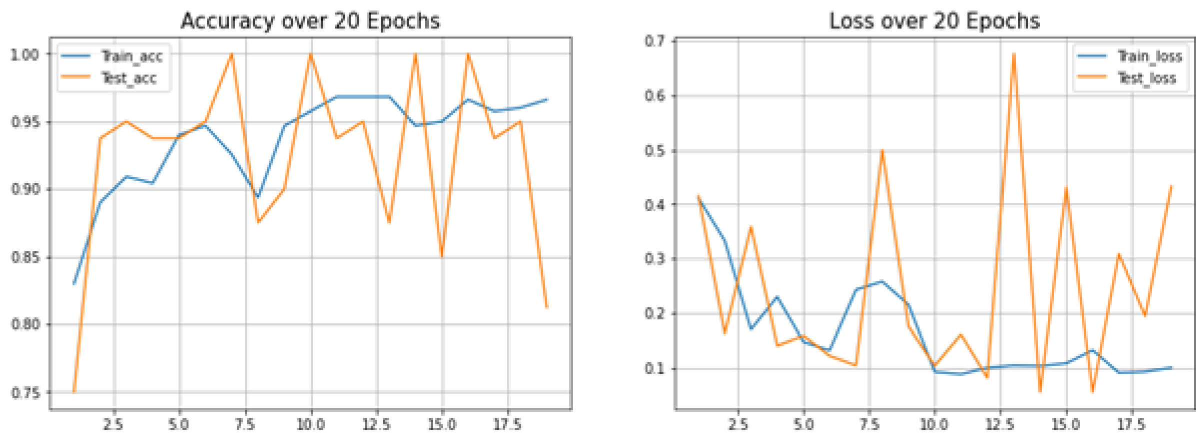
Accuracies and Losses of the last 3 epochs of the binary case.

Finally, we'll use the *matplotlib* library to plot our accuracy and loss parameters as a factor of our training epochs.

```
import matplotlib.pyplot as plt
```

```
      result.history['acc']
loss = result.history['loss']
val_acc = result.history['val_acc']
val_loss = result.history['val_loss']
plt.figure(figsize=(15, 5))
plt.subplot(121)
plt.plot(range(1,epochs), acc[1:], label='Train_acc')
plt.plot(range(1,epochs), val_acc[1:], label='Test_acc')
plt.title('Accuracy over ' + str(epochs) + ' Epochs', size=15)
plt.legend()
plt.grid(True)
plt.subplot(122)
plt.plot(range(1,epochs), loss[1:], label='Train_loss')
plt.plot(range(1,epochs), val_loss[1:], label='Test_loss')
plt.title('Loss over ' + str(epochs) + ' Epochs', size=15)
plt.legend()
plt.grid(True)
plt.show()
```

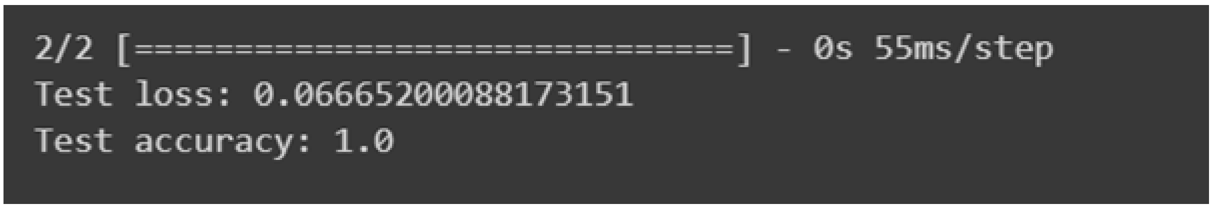
```
plot_acc_loss(result, 20)
```



Accuracy and Loss Parameters, taken over 20 epochs, for the binary case.

To confirm our results, we'll test our model on our test dataset, consisting of 9 images per class. **Note, that you will need to set the *shuffle* parameter to *True* here for accurate results.**

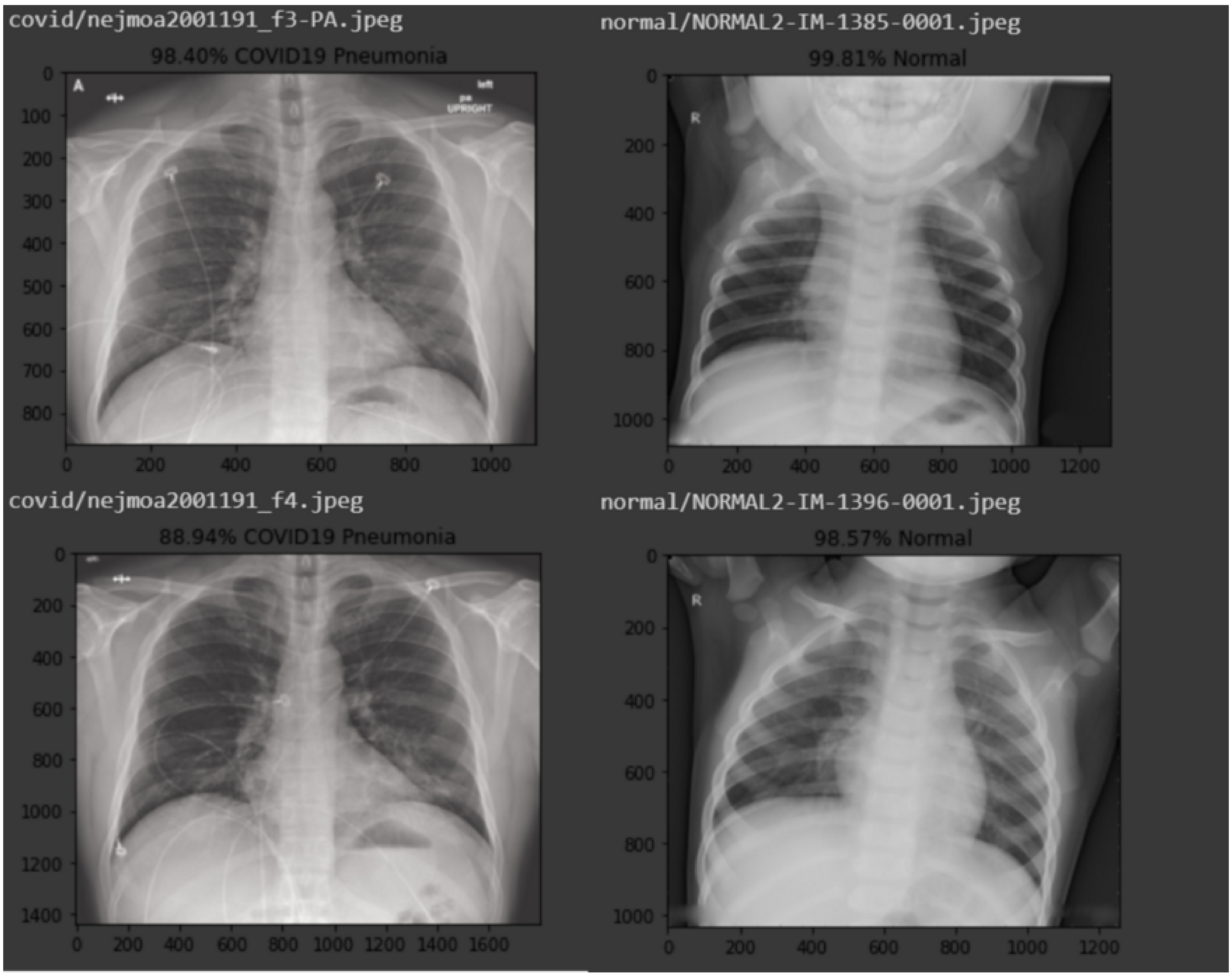
```
test_datagen = ImageDataGenerator(rescale=1. / 255)

eval_generator = test_datagen.flow_from_directory(

 test_dir,target_size=IMAGE_SIZE,
 batch_size=1,
 shuffle=False,
 seed=42,
      enerator.reset()

 x = model.evaluate_generator(eval_generator,

 steps = np.ceil(len(eval_generator) / BATCH_SIZE),
 use_multiprocessing = False,
 verbose = 1,
 workers=1
 )

print('Test loss:' , x[0])
print('Test accuracy:',x[1])
```



```
2/2 [==============================] - 0s 55ms/step
Test loss: 0.06665200088173151
Test accuracy: 1.0
```

While it may seem that our model performed perfecetly for this task, given a much larger test dataset we'd expect this value to match with our validation accuracies. Let's finish by plotting some of our test images along with their respective predictions using the *cv2* image processing class and Keras's *predict_generator()* method. Note that to replicate this, you'll need to set the evaluation_generator's *shuffle* parameter to *False*, to avoid class index shuffling.

```
 eval_generator.reset()
 pred = model.predict_generator(eval_generator,1000,verbose=1)
 print("Predictions finished")

 import cv2

 import matplotlib.image as mpimg
 for index, probability in enumerate(pred):
  image_path = test_dir + "/" +eval_generator.filenames[index]
  image = mpimg.imread(image_path)
  #BGR TO RGB conversion using CV2
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

 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
```


covid/nejmoa2001191_f3-PA.jpeg
98.40% COVID19 Pneumonia
normal/NORMAL2-IM-1385-0001.jpeg
99.81% Normal
covid/nejmoa2001191_f4.jpeg
88.94% COVID19 Pneumonia
normal/NORMAL2-IM-1396-0001.jpeg
98.57% Normal

Prediction probabilities of the COVID19 (Left) and Healthy (Right) test set images.

Our network successfully detected the differences between the two cases. But this is facile, let's up the ante a notch and try discriminating between different types of pneumonia as well. We'll switch over to the second Jupyter notebook, *Covid19_GradientCrescent_Multi.ipynb.*

The Multiclass Case

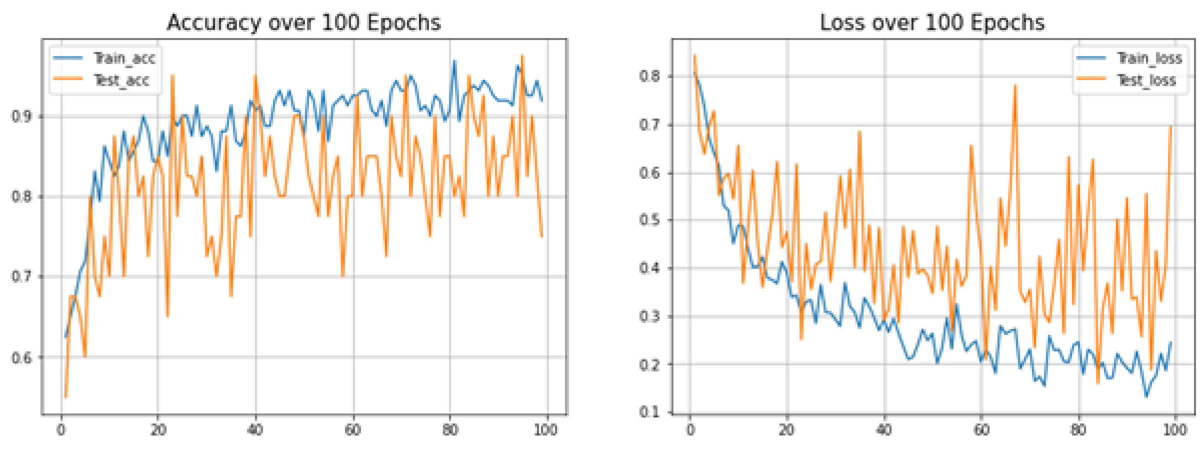
In this case, we'll attempt to not only separate healthy lungs with those infected with pneumonia, but also attempt to discriminate between the different causes of pneumonia, whether they be caused by bacteria, SARS-CoV-2, or some other virus.

Most of our code is identical with that of the binary case. To avoid repetition, let's highlight the key differences:

The number of neurons in our final densely connected layer now corresponds to the number of classes considered, with a *softmax* activation function attached. This will allow us to give a probability output for each of the classes, and to take the maximum as our final predicted output.

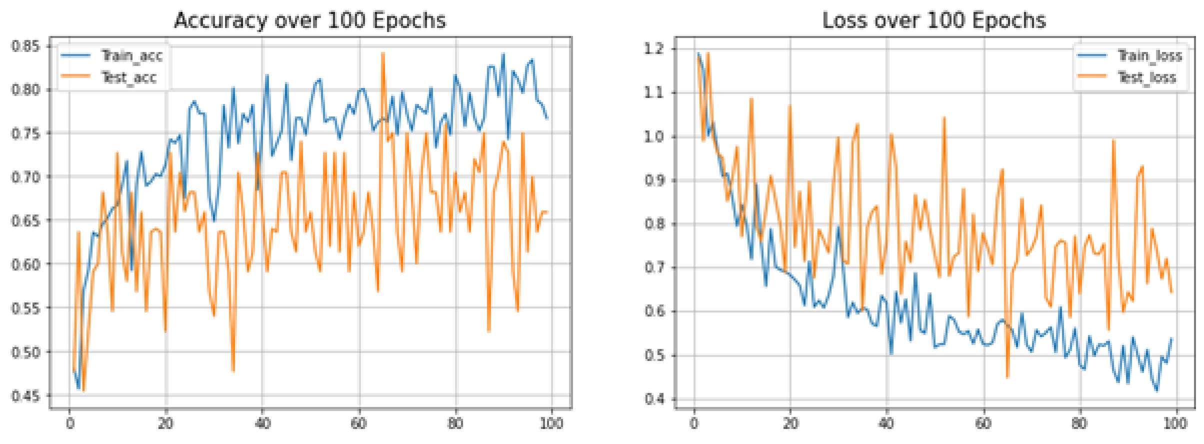
- The model is now compiled with a *categorical_crossentropy* loss function.
- The number of training epochs is increased, to accommodate for the increased situational complexity. Similarly, the learning rate has been decreased to 1E-4.

Let's view the training results when considering three classes (Healthy, COVID-19, Bacterial), after training for 100 epochs at a learning rate of 5E-4.



Accuracy and Loss Parameters, taken over 100 epochs, for the three-class case.

While the performance of our model thus far has been satisfactory, reaching above 80%, let's take a look at the four-class case incorporating other viral causes of pneumonia.



Accuracy and Loss Parameters, taken over 100 epochs, for the four-class case.

caused by the SARS-COV-2 virus, and hence we would expect a strong overlap in the feature domain between these two classes. With a more mesoscopic model, increased data availability, and a longer fine-tuning process, we'd expect the accuracy to increase.

Our results reinforce the difficulty in using CT/X-rays alone in diagnosing COVD-19, and helps to partially explain why health authorities in China abandoned this approach and reverted back to PCR-based testing: with all of the other winter bugs in circulation, it becomes incredibly difficult to pin down the causes of pneumonia in individual cases. It's even possible to have several parallel causes of pneumonia, further complicating clinical diagnosis.

We hope you enjoyed this article, and hope you check out the many other articles on GradientCrescent, covering applied and theoretical aspects of AI. To stay up to date with the latest updates on GradientCrescent, please consider following the publication and following our Github repository.

Remember to stay safe, wash your hands, and to look out for one other. As the Ancient poet Nagaya put it best — "**Though we live at different places, we share the same sky".**