Final Project

CoronaHack -Chest X-Ray-Dataset

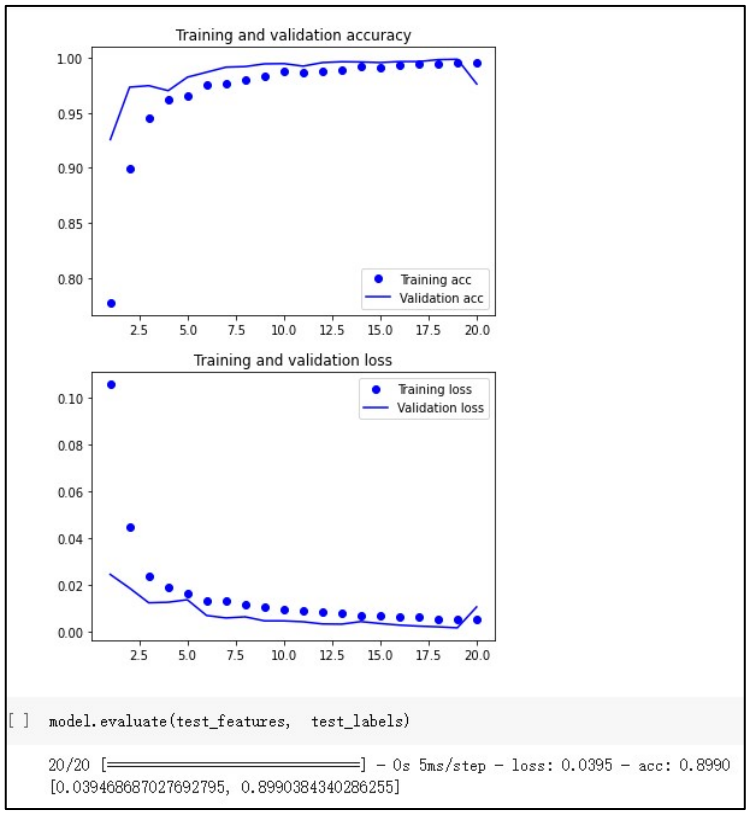
The data was bigger than before. About 1GB. We used to attach a cloud drive. But the teacher said they didn't have an account. We used !gdown to solve this problem.

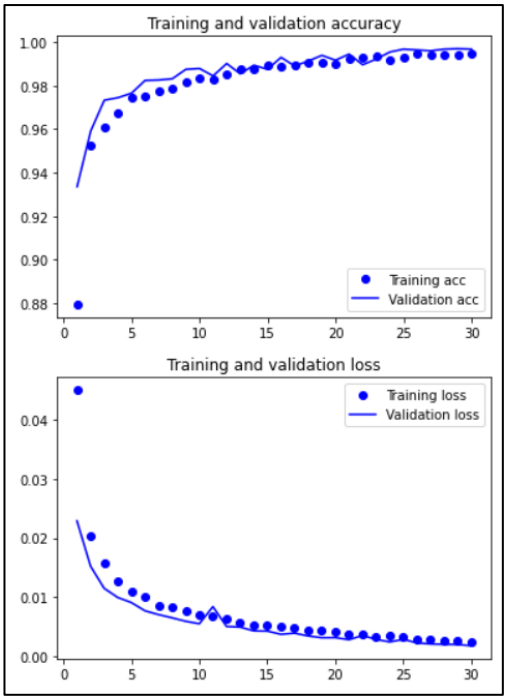
We divided the data to train and test. Find which has a problem or not. The names are Normal and Pneumonia. We assigned the specified folder for each other. Meanwhile, a problem happens. We have no idea how to deal with this. Therefore, it takes a long time to search for a suitable function. It's "iloc". It can search for the folder's exact name. Though the name. We copy the original source to our specified folders.

Next, ImageDataGenerator regularizes data. This is unified data. Let it look tidy. After that, VGG16 is a simple unity. To prove a deeper level and optimize performance.

Compiler! It takes a lot of time. I made a mistake when it worked. Because I modify it while it is in progress. We met a defined problem. Now I learn about why it is wrong. Change to GPU is a tip and tricks. It can save more time.

CBB107004 吳弘濱
CBB107005 黃昱維
CBB107009 吳旻洋
CBA107007 莊季朋





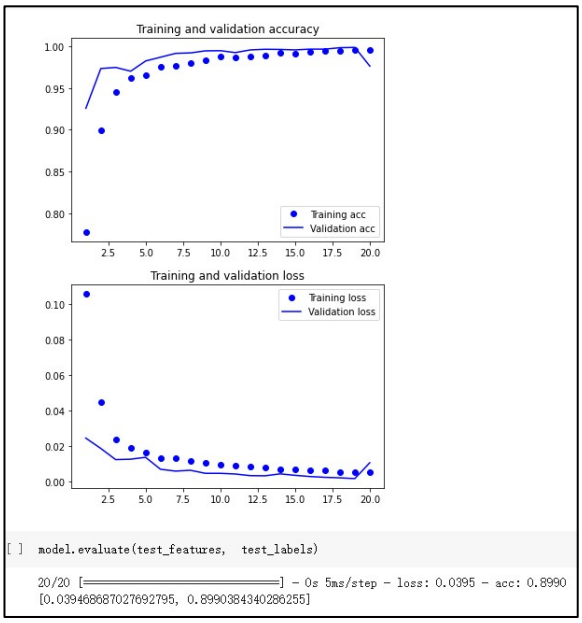
Our test's accuracy is about 77%.

We can see the figure. When we trained 30 times. It's better than starting. And loss less than.

If only we tried too many times. It will cause overfitting. Thus, assigning your data is an important section that we have to decide in the beginning.

```
model.evaluate(test_features,  test_labels)

20/20 [==============================] - 0s 5ms/step - loss: 0.0880 - acc: 0.7788
[0.08804159611463547, 0.7788461446762085]
```



This is the final version. We improve the accuracy. Previous one's accuracy was 77%. The final version figure shows accuracy is 89%. It increased 12%. Moreover, the loss is important too. From 0.088 decrease to 0.0385. That's better now.

```
model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
  loss='binary_crossentropy',
  metrics=['acc'])
```

We modify our original one uses binary's loss function.

```
model.evaluate(test_features,  test_labels)

20/20 [==============================] - 0s 5ms/step - loss: 0.9442 - acc: 0.7500
[0.9442445039749146, 0.75]
```

The evaluation has a high loss.

```
model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
  loss='mean_squared_logarithmic_error',
  metrics=['acc'])
```

So, we try to use other loss functions to decrees loss.

```
[35] model.evaluate(test_features,  test_labels)

     20/20 [==============================] - 0s 5ms/step - loss: 0.0973 - acc: 0.7676
     [0.09728708118200302, 0.7676281929016113]
```

Although we change the loss function to reduce the loss, the accuracy is about 77%. It's not up to snuff. We expect higher than 80%.

```
model.add(layers.Dropout(0.5))
```

We suspect that dropout's rate is too high. Lead to accuracy decreases.

```
model.add(layers.Dropout(0.3))
```

Numerous attempts to test. We modify it to 0.3.

```
[38] model.evaluate(test_features,  test_labels)

     20/20 [==============================] - 0s 5ms/step - loss: 0.0623 - acc: 0.8526
     [0.062343720346689224, 0.8525640964508057]
```

It proved our suspicion that the dropout rate results in accuracy lower than 80%.

```
model.evaluate(test_features,  test_labels)

20/20 [==============================] - 0s 5ms/step - loss: 0.0395 - acc: 0.8990
[0.039468687027692795, 0.8990384340286255]
```

We confirm the truth of the modification and it's the best accuracy that we trained.