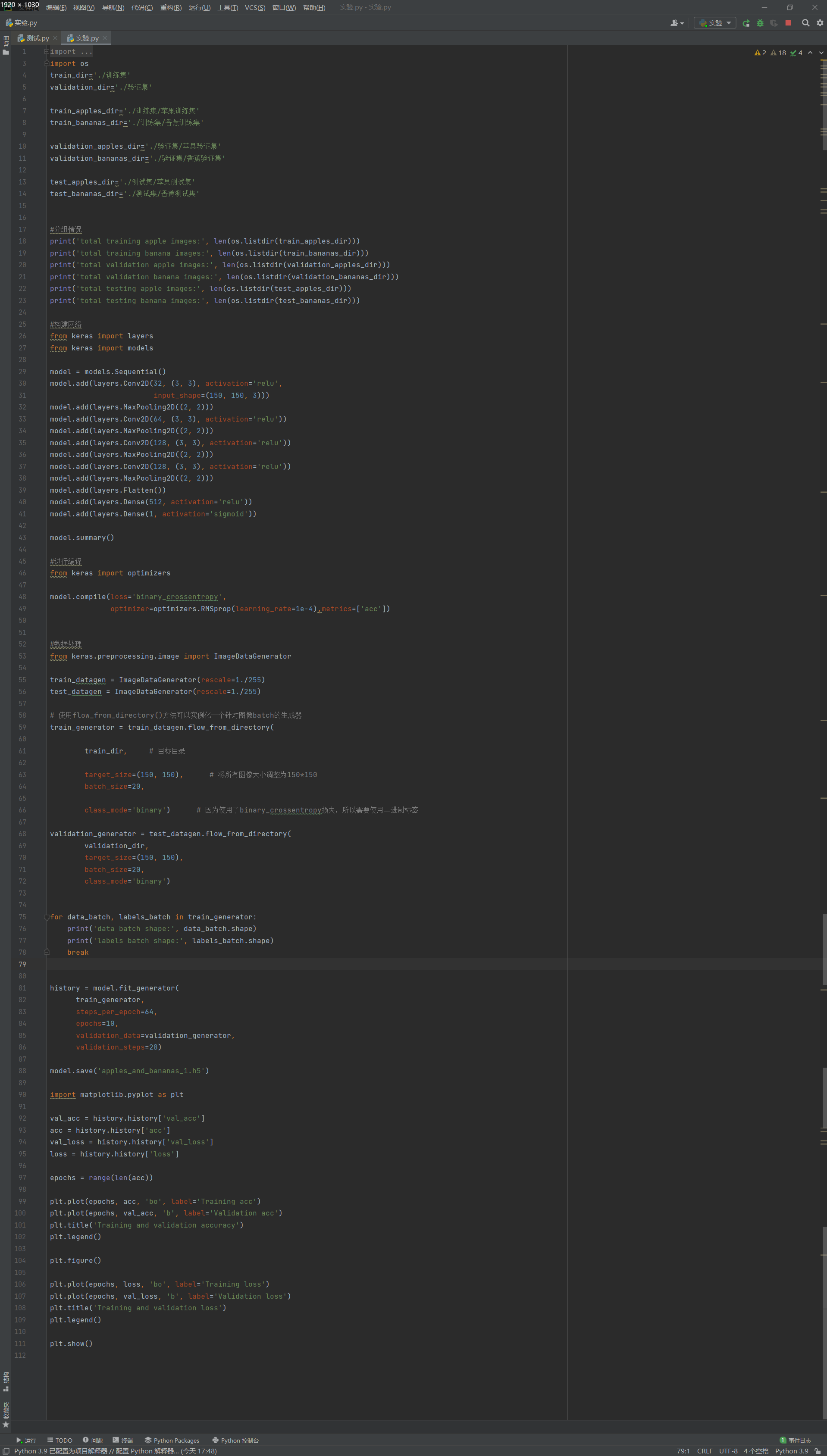
# 基于CNN的水果分类

水果分类任务，本实验使用了最基本的图像二分类技术。

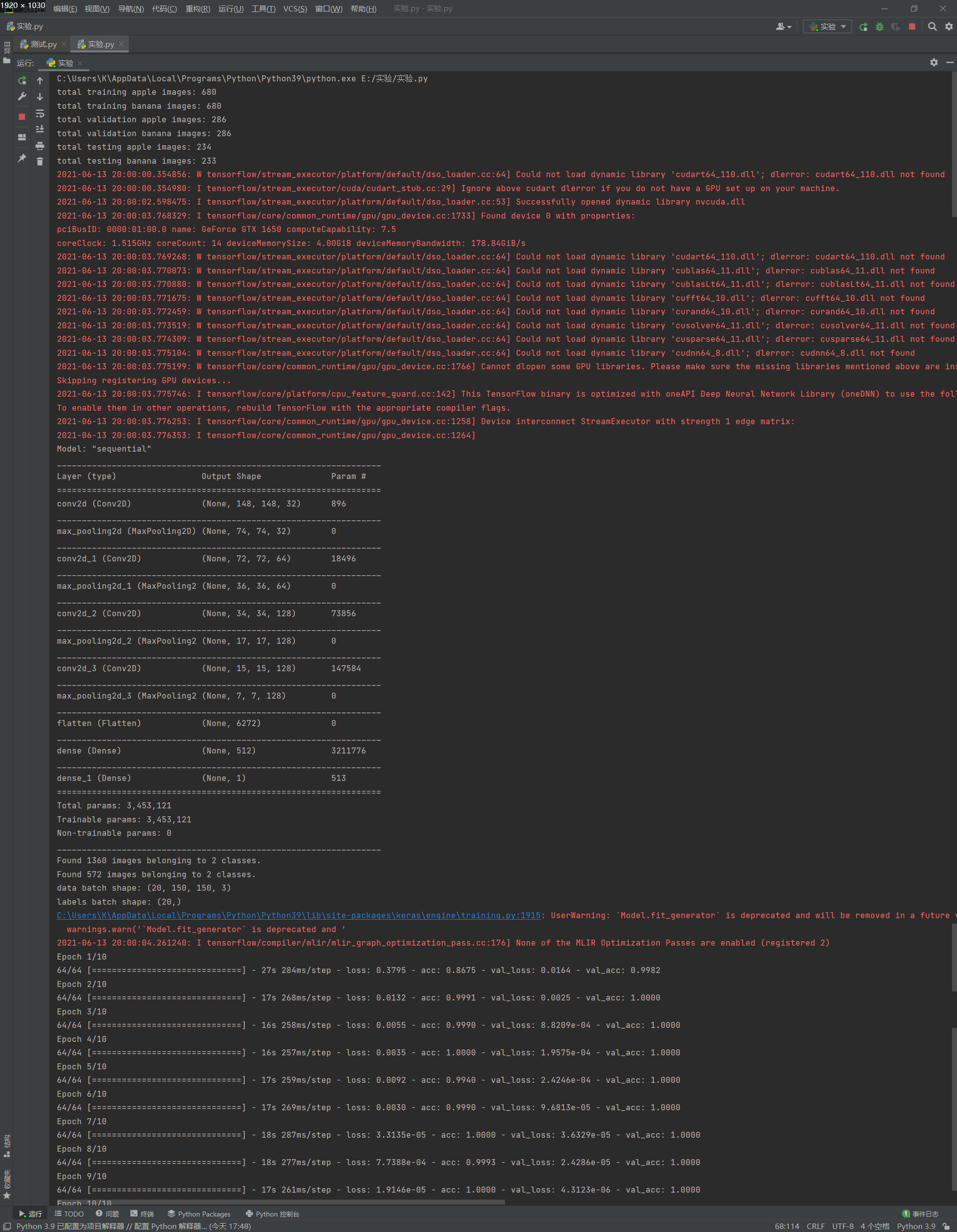
对于这样一个图像分类任务，我们采用了keras库来辅助进行实验。这是因为Keras中的keras.preprocessing.image有包含ImageDataGenerator类，可以快速创建Python生成器，能够将硬盘上的图像文件自动转换为预处理好的张量，很方便进行图像数据的处理。下面是我们的实验代码和实验结果及其说明。

首先放上总体的代码和实验结果。（为了便于阅读，之后的代码和结果不采用截图形式。）

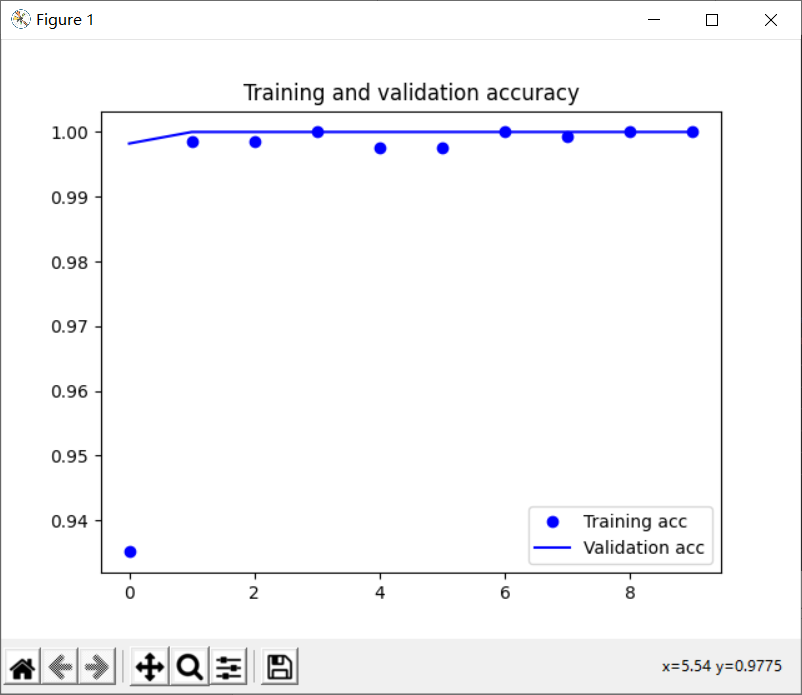
**1、实验代码截图：**

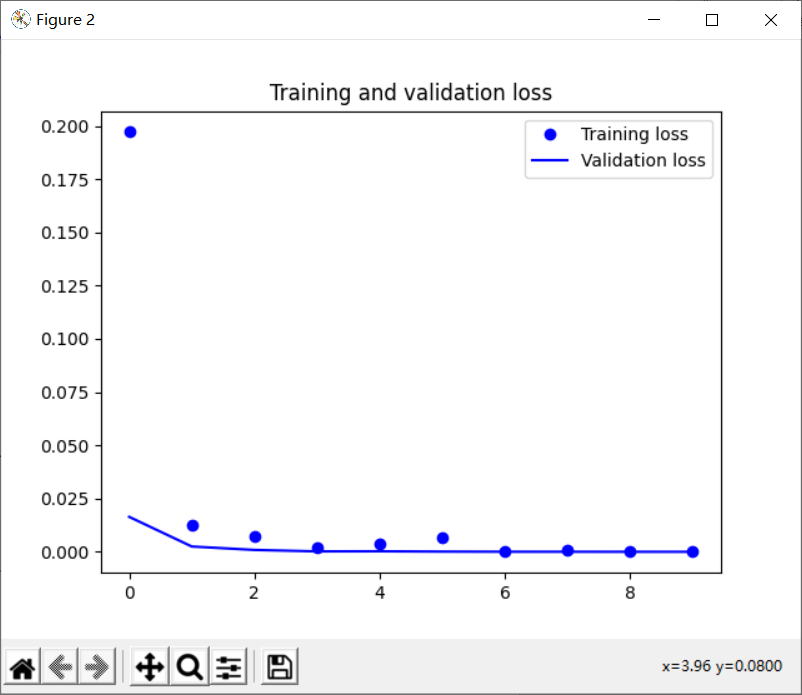


**2、实验结果截图：**



（红色部分是因为我没有安装CUDA，无法使用GPU协助运算，不必理会。）

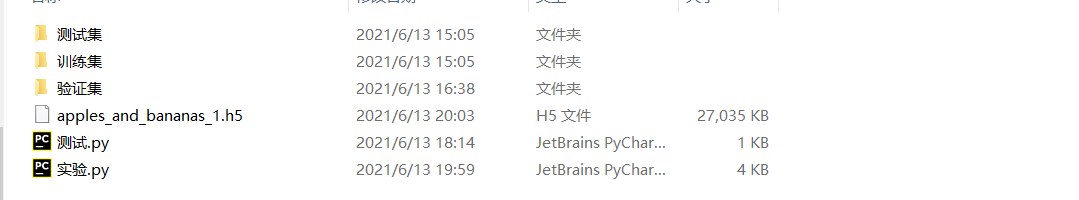




**1 实验代码及其说明**

**1.1准备工作**

我们首先在网上找到了一组关于苹果和香蕉的数据集，并将其分成了训练集、验证集、测试集三部分，如图：



每个部分中又包含苹果和香蕉的图片若干。为了直观的表示每个数据集中包含图片的多少，我们用了课上老师讲的关于文件路径的知识，统计了每个文件夹中的图片个数。代码和结果如下：

**代码：**

1. import os
2. train\_dir='./训练集'
3. validation\_dir='./验证集'
4. train\_apples\_dir='./训练集/苹果训练集'
5. train\_bananas\_dir='./训练集/香蕉训练集'
6. validation\_apples\_dir='./验证集/苹果验证集'
7. validation\_bananas\_dir='./验证集/香蕉验证集'
8. test\_apples\_dir='./测试集/苹果测试集'
9. test\_bananas\_dir='./测试集/香蕉测试集'
10. *#分组情况*
11. print('total training apple images:', len(os.listdir(train\_apples\_dir)))
12. print('total training banana images:', len(os.listdir(train\_bananas\_dir)))
13. print('total validation apple images:', len(os.listdir(validation\_apples\_dir)))
14. print('total validation banana images:', len(os.listdir(validation\_bananas\_dir)))
15. print('total testing apple images:', len(os.listdir(test\_apples\_dir)))
16. print('total testing banana images:', len(os.listdir(test\_bananas\_dir)))

**结果：**

C:\Users\K\AppData\Local\Programs\Python\Python39\python.exe E:/实验/实验.py

total training apple images: 680

total training banana images: 680

total validation apple images: 286

total validation banana images: 286

total testing apple images: 234

total testing banana images: 233

**1.2构建网络**

1. *#构建网络*
2. from keras import layers
3. from keras import models
4. model = models.Sequential()
5. model.add(layers.Conv2D(32, (3, 3), activation='relu',
6. input\_shape=(150, 150, 3)))
7. model.add(layers.MaxPooling2D((2, 2)))
8. model.add(layers.Conv2D(64, (3, 3), activation='relu'))
9. model.add(layers.MaxPooling2D((2, 2)))
10. model.add(layers.Conv2D(128, (3, 3), activation='relu'))
11. model.add(layers.MaxPooling2D((2, 2)))
12. model.add(layers.Conv2D(128, (3, 3), activation='relu'))
13. model.add(layers.MaxPooling2D((2, 2)))
14. model.add(layers.Flatten())
15. model.add(layers.Dense(512, activation='relu'))
16. model.add(layers.Dense(1, activation='sigmoid'))
17. model.summary()

这里，我们采用的是conv2D和MaxPooling2D交替组合的形式。特征图的深度在逐渐增加（从32增加到128），而特征图的尺寸在逐渐减小（从150150减小到77），这几乎是所有卷积神经网络的模式。这样做还有一个好处是最后使用Flatten的时候尺寸不会太大。

因为，猫狗识别任务是一个二分类任务，所以网络最后一层使用的是sigmoid激活的单一单元（大小为1的Dense层）对某个类别的概率进行编码。

输出结果如下：

1. Layer (type)                 Output Shape              Param *#*
2. =================================================================
3. conv2d\_1 (Conv2D)            (None, 148, 148, 32)      896
4. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. max\_pooling2d\_1 (MaxPooling2 (None, 74, 74, 32)        0
6. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. conv2d\_2 (Conv2D)            (None, 72, 72, 64)        18496
8. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
9. max\_pooling2d\_2 (MaxPooling2 (None, 36, 36, 64)        0
10. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
11. conv2d\_3 (Conv2D)            (None, 34, 34, 128)       73856
12. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
13. max\_pooling2d\_3 (MaxPooling2 (None, 17, 17, 128)       0
14. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
15. conv2d\_4 (Conv2D)            (None, 15, 15, 128)       147584
16. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
17. max\_pooling2d\_4 (MaxPooling2 (None, 7, 7, 128)         0
18. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
19. flatten\_1 (Flatten)          (None, 6272)              0
20. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
21. dense\_1 (Dense)              (None, 512)               3211776
22. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
23. dense\_2 (Dense)              (None, 1)                 513
24. =================================================================
25. Total params: 3,453,121
26. Trainable params: 3,453,121
27. Non-trainable params: 0
28. Found 1360 images belonging to 2 classes.
29. Found 572 images belonging to 2 classes.

下面进行编译：

1. from keras import optimizers
2. model.compile(loss='binary\_crossentropy',
3. optimizer=optimizers.RMSprop(lr=1e-4),
4. metrics=['acc'])

**1.3数据处理**

由于图片不能直接放入神经网络中进行学习，学习之前应该把数据格式化为经过预处理的浮点数张量。所以我们将JPG图像文件解码为RGB像素网格，然后将这些像素网格转换成浮点数张量，最后将像素值（0-255范围内）缩放到[0,1]区间，因为神经网络处理较小的数据输入值较快。下面是我们的代码：

1. *#数据处理*
2. from keras.preprocessing.image import ImageDataGenerator
3. train\_datagen = ImageDataGenerator(rescale=1./255)
4. test\_datagen = ImageDataGenerator(rescale=1./255)
5. *# 使用flow\_from\_directory()方法可以实例化一个针对图像batch的生成器*
6. train\_generator = train\_datagen.flow\_from\_directory(
8. train\_dir,     *# 目标目录*
10. target\_size=(150, 150),      *# 将所有图像大小调整为150\*150*
11. batch\_size=20,
13. class\_mode='binary')      *# 因为使用了binary\_crossentropy损失，所以需要使用二进制标签*
14. validation\_generator = test\_datagen.flow\_from\_directory(
15. validation\_dir,
16. target\_size=(150, 150),
17. batch\_size=20,
18. class\_mode='binary')

来看一下其中一个生成器的输出：它生成150\*150的RGB图像[形状为（20， 150， 150， 3）]与二进制标签[形状为（20，）]组成的批量。每个批量包含20个样本。由于生成器会不停的生成这种批量，它会不断的循环目标文件夹中的图像，因此需要在某个时刻break迭代循环。

1. for data\_batch, labels\_batch in train\_generator:
2. print('data batch shape:', data\_batch.shape)
3. print('labels batch shape:', labels\_batch.shape)
4. break

输出为：

1. data batch shape: (20, 150, 150, 3)
2. labels batch shape: (20,)

**1.4模型拟合**

接下来我们让模型对数据进行拟合。

对于model.fit\_generator中的参数：里面含有上面代码的两个生成器train\_generator和validation\_generator；训练集样本总数是1360，我们设置的batch\_size为20，因此steps\_per\_epoc取值至多为1360/20=68，我们找了一个比较好看的数64作为steps\_per\_epoc的值；验证集样本总数为572，因此validation\_steps取值为572//20=28。

参数设置好后的代码如下：

1. history = model.fit\_generator(
2. train\_generator,
3. steps\_per\_epoch=64,
4. epochs=10,
5. validation\_data=validation\_generator,
6. validation\_steps=28)

最后我们把模型存储下来，运行一下：

1. model.save('apples\_and\_bananas\_1.h5')

**结果：**

1. Epoch 1/10
2. 64/64 [==============================] - 27s 284ms/step - loss: 0.3795 - acc: 0.8675 - val\_loss: 0.0164 - val\_acc: 0.9982
3. Epoch 2/10
4. 64/64 [==============================] - 17s 268ms/step - loss: 0.0132 - acc: 0.9991 - val\_loss: 0.0025 - val\_acc: 1.0000
5. Epoch 3/10
6. 64/64 [==============================] - 16s 258ms/step - loss: 0.0055 - acc: 0.9990 - val\_loss: 8.8209e-04 - val\_acc: 1.0000
7. Epoch 4/10
8. 64/64 [==============================] - 16s 257ms/step - loss: 0.0035 - acc: 1.0000 - val\_loss: 1.9575e-04 - val\_acc: 1.0000
9. Epoch 5/10
10. 64/64 [==============================] - 17s 259ms/step - loss: 0.0092 - acc: 0.9940 - val\_loss: 2.4246e-04 - val\_acc: 1.0000
11. Epoch 6/10
12. 64/64 [==============================] - 17s 269ms/step - loss: 0.0030 - acc: 0.9990 - val\_loss: 9.6813e-05 - val\_acc: 1.0000
13. Epoch 7/10
14. 64/64 [==============================] - 18s 287ms/step - loss: 3.3135e-05 - acc: 1.0000 - val\_loss: 3.6329e-05 - val\_acc: 1.0000
15. Epoch 8/10
16. 64/64 [==============================] - 18s 277ms/step - loss: 7.7388e-04 - acc: 0.9993 - val\_loss: 2.4286e-05 - val\_acc: 1.0000
17. Epoch 9/10
18. 64/64 [==============================] - 17s 261ms/step - loss: 1.9146e-05 - acc: 1.0000 - val\_loss: 4.3123e-06 - val\_acc: 1.0000
19. Epoch 10/10
20. 64/64 [==============================] - 17s 266ms/step - loss: 2.1675e-05 - acc: 1.0000 - val\_loss: 9.7942e-07 - val\_acc: 1.0000

大功告成！

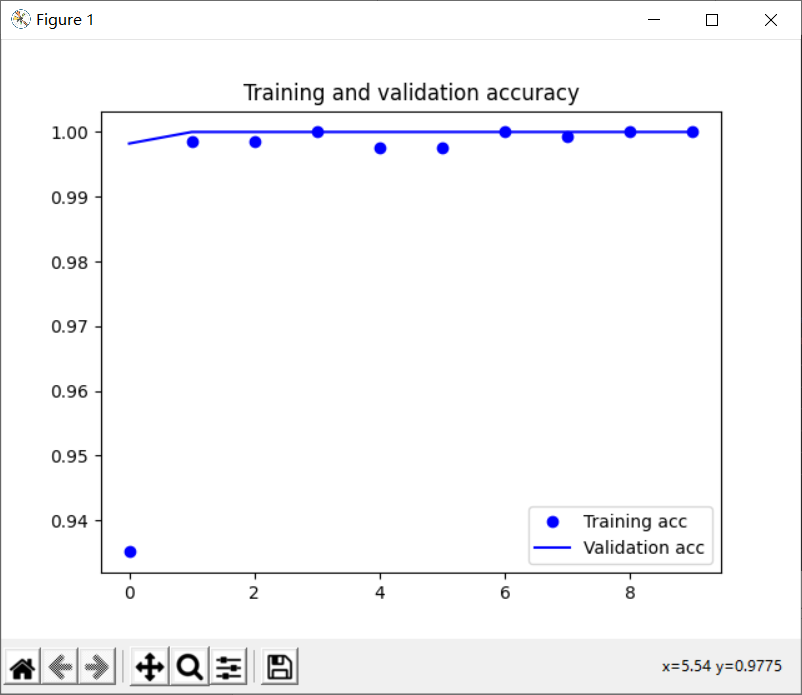
**2 绘制模型在训练数据和验证数据的损失函数和精度**

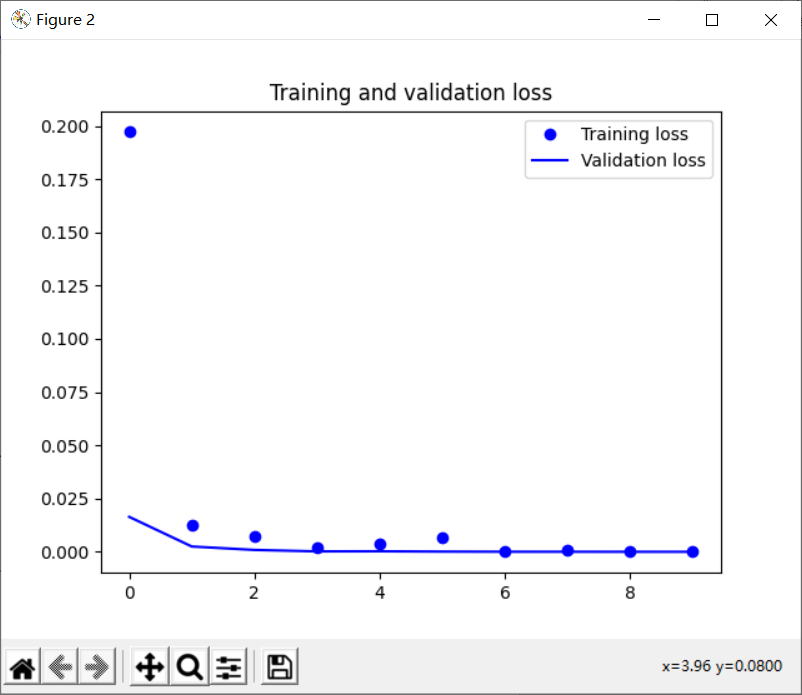
我们使用了老师在课上讲的matplotlib库进行训练过程中模型在训练数据和验证数据的损失函数和精度的可视化。代码如下：

1. import matplotlib.pyplot as plt
2. val\_acc = history.history['val\_acc']
3. acc = history.history['acc']
4. val\_loss = history.history['val\_loss']
5. loss = history.history['loss']
6. epochs = range(len(acc))
7. plt.plot(epochs, acc, 'bo', label='Training acc')
8. plt.plot(epochs, val\_acc, 'b', label='Validation acc')
9. plt.title('Training and validation accuracy')
10. plt.legend()
11. plt.figure()
12. plt.plot(epochs, loss, 'bo', label='Training loss')
13. plt.plot(epochs, val\_loss, 'b', label='Validation loss')
14. plt.title('Training and validation loss')
15. plt.legend()
16. plt.show()

我们使用.history()方法提取每次训练中训练数据的损失函数和精度以及验证数据的损失函数和精度，作为坐标数据的来源。其中val\_acc和val\_loss分别表示验证集的准确度和损失函数；acc和loss分别表示训练集的准确度和损失函数。

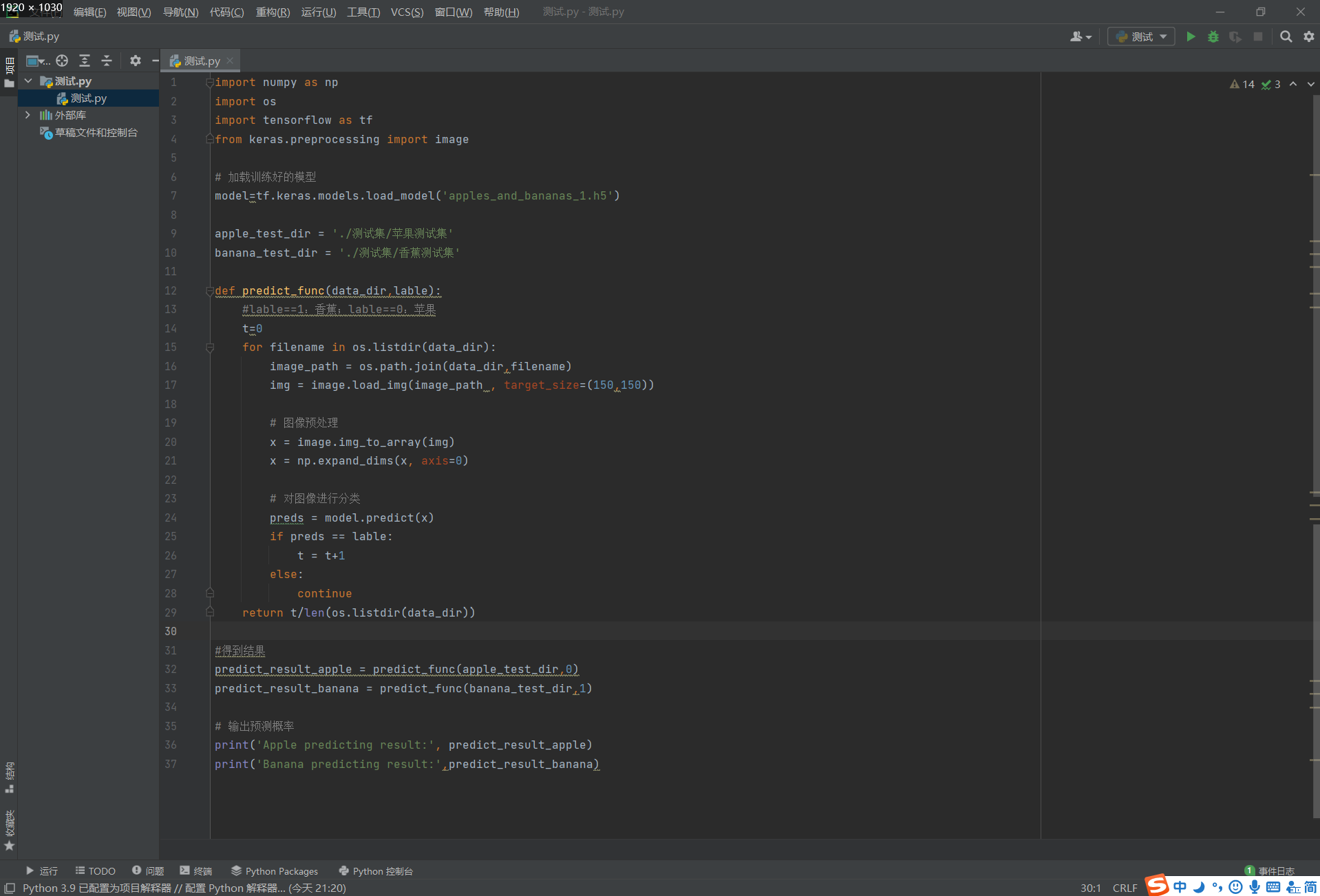
可视化结果如下：





**3 测试模型**

代码截图：



将模型保存后，我进行了模型的测试。引用了老师上课讲的os库进行图片路径的获取；引用TensorFlow库加载模型；引用keras库进行图片处理；引用numpy库进行数据预处理。具体的代码如下：

1. import numpy as np
2. import os
3. import tensorflow as tf
4. from keras.preprocessing import image
5. *# 加载训练好的模型*
6. model=tf.keras.models.load\_model('apples\_and\_bananas\_1.h5')
7. apple\_test\_dir = './测试集/苹果测试集'
8. banana\_test\_dir = './测试集/香蕉测试集'
9. def predict\_func(data\_dir,lable):
10. *#lable==1：香蕉；lable==0：苹果*
11. t=0
12. for filename in os.listdir(data\_dir):
13. image\_path = os.path.join(data\_dir,filename)
14. img = image.load\_img(image\_path , target\_size=(150,150))
15. *# 图像预处理*
16. x = image.img\_to\_array(img)
17. x = np.expand\_dims(x, axis=0)
18. *# 对图像进行分类*
19. preds = model.predict(x)
20. if preds == lable:
21. t = t+1
22. else:
23. continue
24. return t/len(os.listdir(data\_dir))
25. *#得到结果*
26. predict\_result\_apple = predict\_func(apple\_test\_dir,0)
27. predict\_result\_banana = predict\_func(banana\_test\_dir,1)
28. *# 输出模型预测准确率*
29. print('Apple predicting result:', predict\_result\_apple)
30. print('Banana predicting result:',predict\_result\_banana)

结果如下：

1. Apple predicting result: 1.0
2. Banana predicting result: 1.0

非常准确！

**4 实验反思**

准确率很高的原因是我们找到的水果样本图片过于纯粹，除了水果本身之外几乎没有其他因素的干扰，而测试集合并没有更多的干扰因素。



如果使用驳杂的训练样本，那么训练出来的模型预测时的准确率就会有所下降，不会是惊人的接近100%。在数据收集时应注意。