Purpose

In this experiment, we will complete the classification and recognition of fruits. Our experiment mainly lies in the establishment of the early model. After the model is established, if it can be further developed and applied to real life, I think there are several potential areas that can be used. ① Intelligent learning program for infants. We can help them know the name of the fruit. ② Supermarket's self checkout program. Used to identify the type of goods, and play the same role as the bar code.

Because the application part of the model needs skilled knowledge about the development of front-end, so this experiment mainly focuses on the establishment of the model. After that, we may further learn the development knowledge and realize the function of the model.

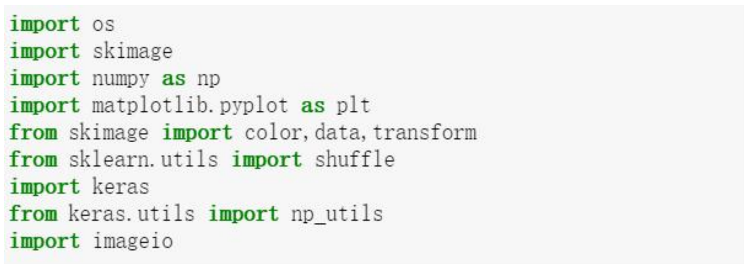
Process

This experiment is mainly divided into the following steps: ① Data acquisition; ② Data visualization; ③ Data preprocessing; ④ Building tensorflow to build convolutional neural network model; ⑤ Selection and effect comparison of optimizer; ⑥ Checking the accuracy of the model on the test set; ⑦ Using OpenCV to visualize the model

① Data acquisition

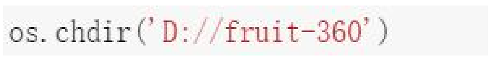
The data set used in this experiment is Fruit-360. The dataset contains 90483 images, including 67692 images in the training set and 22688 images in the test set. The size of each image is 100x100 pixels. A link to the dataset can be found in the appendix.

Load the library functions needed in the experiment.



```python
import os
import skimage
import numpy as np
import matplotlib.pyplot as plt
from skimage import color,data,transform
from sklearn.utils import shuffle
import keras
from keras.utils import np_utils
import imageio
```

Read file path.

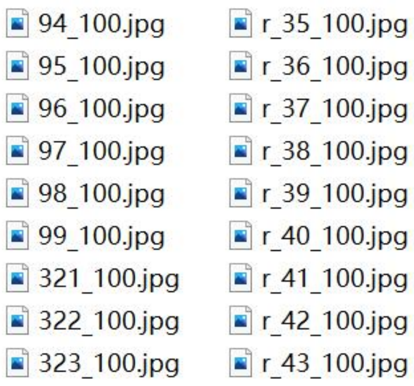


```python
os.chdir('D://fruit-360')
```

Next, we use the **os. listdir()** function to realize the circular reading of the graph under the folder. The returned data includes all data and its labels. Because the folder is a string, it is not convenient for the later hot code conversion, so it is stored as an iteration of the outer loop. The fifth picture under each folder and its corresponding folder name are used as labels to visualize the 5th picture of each kind of fruit.

```
def load_data(dir_path):
    images=[] ##Store pictures
    labels=[] ##Store lables
    no5_imgs=[] ##Store the 5th picture in each folder
    labels_no5=[] ##Store the 5th lables in each folder
    lab=os.listdir(dir_path)
    n=0
    for l in lab:
        img=os.listdir(dir_path+l)
        for i in img:
            img_path=dir_path+l+'/'+i
            labels.append(int(n))
            images.append(imageio.imread(img_path))
        n+=1
        no5_img=format_path(img) ##Arrange the pictures in the correct order to get the fifth picture
        img5_path=dir_path+l+'/'+no5_img
        labels_no5.append(l)
        no5_imgs.append(imageio.imread(img5_path)) ##The fifth picture of each category is read out and stored in the no5_imgs
    return images,labels,no5_imgs,labels_no5
```

After carefully observing the folder where the pictures are stored, we can find that the names of the pictures are roughly divided into two categories. One is that the first part of the file name can be converted to integer, the other is that the file name can not be converted to integer after the file name is split. As shown in the figure below.


94_100.jpg
95_100.jpg
96_100.jpg
97_100.jpg
98_100.jpg
99_100.jpg
321_100.jpg
322_100.jpg
323_100.jpg
r_35_100.jpg
r_36_100.jpg
r_37_100.jpg
r_38_100.jpg
r_39_100.jpg
r_40_100.jpg
r_41_100.jpg
r_42_100.jpg
r_43_100.jpg

In order to visualize the fifth picture, we need to define the following functions to arrange the labels of the picture.

```
##Function to read files in folder in order
def format_path(img):
    yes_int=[] ##New list is used to store the first file name that can be converted to an integer after segmentation
    for s in range(len(img)):
        img[s] = img[s].split('_') ##Separate file names with'_'
        if(is_number(img[s][0])): ##Determine whether the first part of the file name after segmentation can be converted to integer
            img[s][0]=int(img[s][0]) ##Convert the first part of the file name to an integer
            yes_int.append(img[s]) ##Put the file name in the list without 'r'
    yes_int.sort() ##sort
    for yi in range(len(yes_int)): #
        yes_int[yi][0]=str(yes_int[yi][0]) ##convert to str
        yes_int[yi]=yes_int[yi][0]+'_'+yes_int[yi][1] ##joint
    no5_img=yes_int[4]
    return no5_img
```

```
##Judge whether the data can be converted to plastic
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        pass

    try:
        import unicodedata
        unicodedata.numeric(s)
        return True
    except (TypeError, ValueError):
        pass

    return False
```

The following functions are used to load test set and training set data.

```
##Training
images,labels,no5_imgs,labels_no5=load_data('.\Training\\')
print(len(images),len(labels),len(no5_imgs))
```

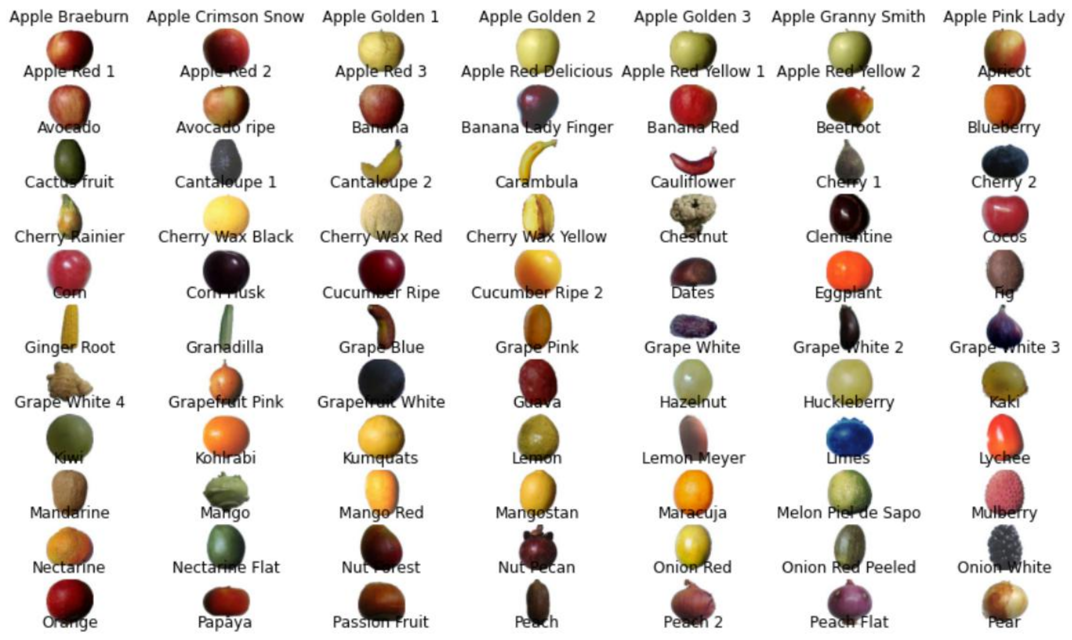
```
##Test
images,labels,no5_imgs,labels_no5=load_data('.\Test\\')
print(len(images),len(labels),len(no5_imgs))
```

② Data visualization

Next, we visualize the data. Define a function to traverse no5_ imgs data set, display the pictures inside and the corresponding labels_ no5。

```
##Function
def display_no5_img(no5_imgs,labels_no5):
    fig = plt.figure(figsize=(15,15)) ##size
    for i in range(len(no5_imgs)):
        plt.subplot(19,7,(i+1)) ##10 pictures per line, 14 lines in total
        plt.title("{0}".format(labels_no5[i])) ##lables
        plt.imshow(no5_imgs[i])  ##pictures
        plt.axis('off')
##Display
display_no5_img(no5_imgs,labels_no5)
```

The following is a visualization of the data.



③ Data preprocessing

Because there are more than 60000 pictures of 131 kinds of fruits that need to be trained, my equipment can't afford such a large amount of data. Therefore, in this experiment, we selected the top 20 kinds of fruits to establish the model. If you want to expand the amount of data, you only need to modify the number of categories. We use the following function to load a small dataset.

```
##n represents the number of categories
def load_small_data(dir_path,m):
    images_m=[]
    labels_m=[]
    lab=os.listdir(dir_path)
    n=0
    for l in lab:
        if(n>=m):
            break
        img=os.listdir(dir_path+l)
        for i in img:
            img_path=dir_path+l+'/'+i
            labels_m.append(int(n))
            images_m.append(imageio.imread(img_path))
        n+=1
    return images_m,labels_m
```

We set the category to 20, and load the training set and test set.

```
images_20,labels_20=load_small_data('.\Training\\',20) ##Training
images_test_20,labels_test_20=load_small_data('.\Test\\',20) ##Test
```

In image processing, there are many ways, such as image denoising, translation, inversion, graying, clipping and so on. I used the image data set after gray processing to train the model, and the accuracy of the test set is reduced. Because for the identification of fruit species, color is also an important feature to distinguish fruits. Therefore, in this experiment, I mainly carried out the batch cutting of images, and converted the image data set into array form, disordered it, and carried out the unique hot coding of keras on the transition data set. The unique hot coding can deal with discontinuous numerical features, and it is actually an extension of the features. The following is the image preprocessing function.

```
def cut_image(images,w,h):
    new_images=[skimage.transform.resize(I,(w,h)) for I in images]
    return new_images
def prepare_data(images,labels,n_classes):
    images64=cut_image(images,64,64) ##cut images
    train_x=np.array(images)
    train_y=np.array(labels)
    indx=np.arange(0,train_y.shape[0])
    indx=shuffle(indx)
    train_x=train_x[indx]
    train_y=train_y[indx]
    train_y=keras.utils.to_categorical(train_y,n_classes) ##one-hot one-hot encoding
    return train_x,train_y
```

The data of training set and test set are preprocessed.

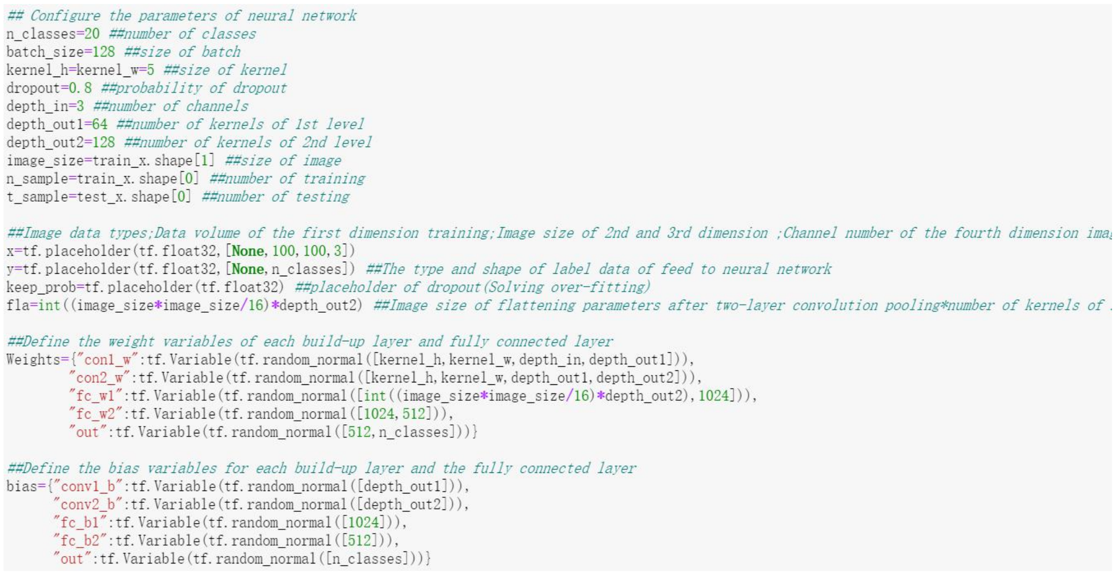
```
##training data
train_x,train_y=prepare_data(images_20,labels_20,20)
##test data
test_x,test_y=prepare_data(images_test_20,labels_test_20,20)
```

④ Building tensorflow to build convolutional neural network model

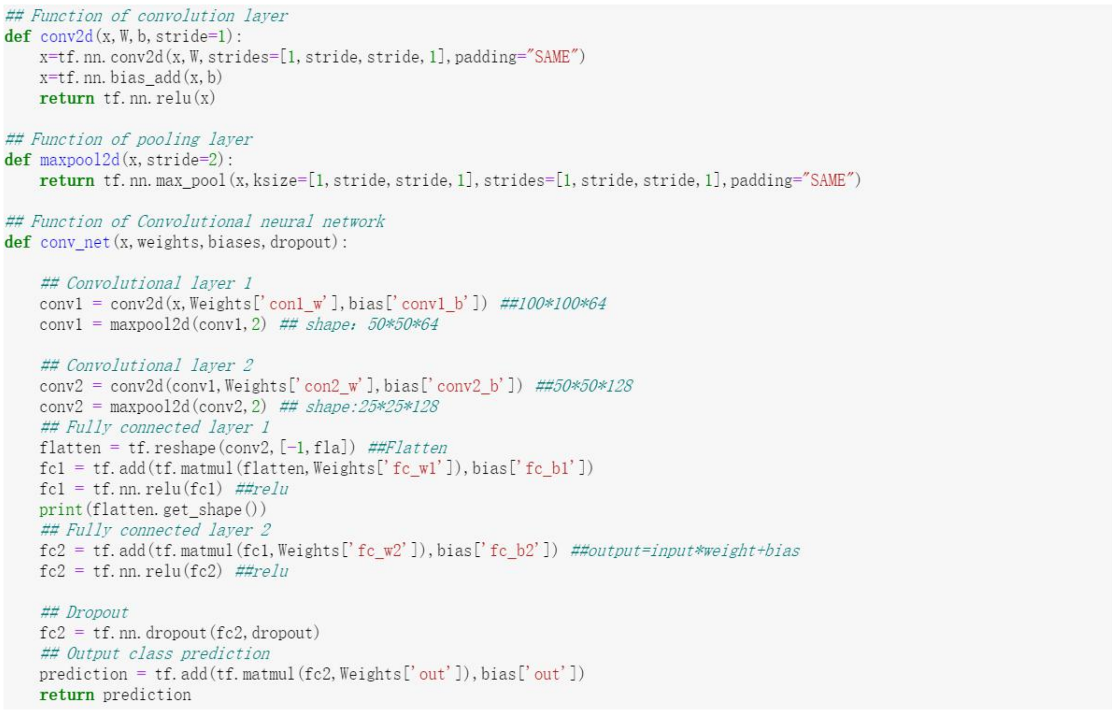
This experiment uses the classical model LeNet-5 model to build two-layer convolution pooling, three-layer full connection.

LeNet-5 network transforms the original image into a series of feature images by alternately connecting the convolution layer and the down sampling layer, and transfers these features to the fully connected neural network to classify the image according to the characteristics of the image. Its biggest advantage lies in its simple structure and easy to understand.

About the construction of LeNet-5 network, the specific comments are in the following code.



```
## Configure the parameters of neural network
n_classes=20 ##number of classes
batch_size=128 ##size of batch
kernel_h=kernel_w=5 ##size of kernel
dropout=0.8 ##probability of dropout
depth_in=3 ##number of channels
depth_out1=64 ##number of kernels of 1st level
depth_out2=128 ##number of kernels of 2nd level
image_size=train_x.shape[1] ##size of image
n_sample=train_x.shape[0] ##number of training
t_sample=test_x.shape[0] ##number of testing

##Image data types;Data volume of the first dimension training;Image size of 2nd and 3rd dimension ;Channel number of the fourth dimension ima
x=tf.placeholder(tf.float32,[None,100,100,3])
y=tf.placeholder(tf.float32,[None,n_classes]) ##The type and shape of label data of feed to neural network
keep_prob=tf.placeholder(tf.float32) ##placeholder of dropout(Solving over-fitting)
fla=int((image_size*image_size/16)*depth_out2) ##Image size of flattening parameters after two-layer convolution pooling*number of kernels of

##Define the weight variables of each build-up layer and fully connected layer
Weights={"con1_w":tf.Variable(tf.random_normal([kernel_h,kernel_w,depth_in,depth_out1])),
        "con2_w":tf.Variable(tf.random_normal([kernel_h,kernel_w,depth_out1,depth_out2])),
        "fc_w1":tf.Variable(tf.random_normal([int((image_size*image_size/16)*depth_out2),1024])),
        "fc_w2":tf.Variable(tf.random_normal([1024,512])),
        "out":tf.Variable(tf.random_normal([512,n_classes]))}

##Define the bias variables for each build-up layer and the fully connected layer
bias={"conv1_b":tf.Variable(tf.random_normal([depth_out1])),
      "conv2_b":tf.Variable(tf.random_normal([depth_out2])),
      "fc_b1":tf.Variable(tf.random_normal([1024])),
      "fc_b2":tf.Variable(tf.random_normal([512])),
      "out":tf.Variable(tf.random_normal([n_classes]))}
```



```
## Function of convolution layer
def conv2d(x,W,b,stride=1):
    x=tf.nn.conv2d(x,W,strides=[1,stride,stride,1],padding="SAME")
    x=tf.nn.bias_add(x,b)
    return tf.nn.relu(x)

## Function of pooling layer
def maxpool2d(x,stride=2):
    return tf.nn.max_pool(x,ksize=[1,stride,stride,1],strides=[1,stride,stride,1],padding="SAME")

## Function of Convolutional neural network
def conv_net(x,weights,biases,dropout):

    ## Convolutional layer 1
    conv1 = conv2d(x,Weights['con1_w'],bias['conv1_b']) ##100*100*64
    conv1 = maxpool2d(conv1,2) ## shape: 50*50*64

    ## Convolutional layer 2
    conv2 = conv2d(conv1,Weights['con2_w'],bias['conv2_b']) ##50*50*128
    conv2 = maxpool2d(conv2,2) ## shape:25*25*128
    ## Fully connected layer 1
    flatten = tf.reshape(conv2,[-1,fla]) ##Flatten
    fc1 = tf.add(tf.matmul(flatten,Weights['fc_w1']),bias['fc_b1'])
    fc1 = tf.nn.relu(fc1) ##relu
    print(flatten.get_shape())
    ## Fully connected layer 2
    fc2 = tf.add(tf.matmul(fc1,Weights['fc_w2']),bias['fc_b2']) ##output=input*weight+bias
    fc2 = tf.nn.relu(fc2) ##relu

    ## Dropout
    fc2 = tf.nn.dropout(fc2,dropout)
    ## Output class prediction
    prediction = tf.add(tf.matmul(fc2,Weights['out']),bias['out'])
    return prediction
```

The theoretical knowledge of neural network is not described too much. The code is available in the appendix.

⑤ Selection and effect comparison of optimizer

In this experiment, we selected the following optimizers.

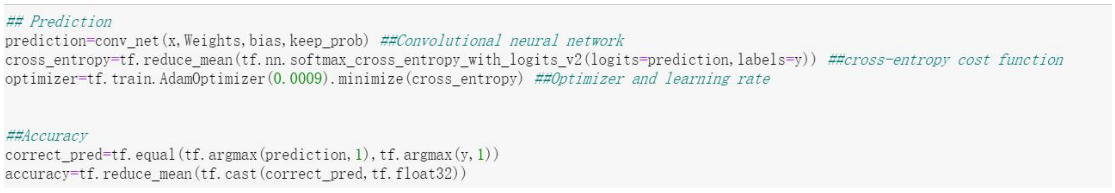
	Learning rate	Number of training	The accuracy of training set	The accuracy of test set
tf.train.GradientDescentOptimizer	0.01	2	very low	<0.1
tf.train.AdagraOptimizer	0.01	2	0.9333	0.85
tf.train.AdamOptimizer	0.01	2	≈1.0	0.9

Among them, **tf.train.GradientDescentOptimizer** training set accuracy is very low, and there will be data loss in the process of training. **tf.train.AdagraOptimizer** can automatically adjust the learning rate, and its training set accuracy growth rate will decrease with the increase of the number of iterations, but the disadvantage is that the time is too long. In contrast, **tf. train. AdamOptimizer** can guarantee accuracy and save time. So, in the next experiment, we choose **tf. train. AdamOptimizer.**

In addition, we also compare the performance of **tf.train.AdamOptimizer** under different learning rates and training times, and the results are shown in the following table.

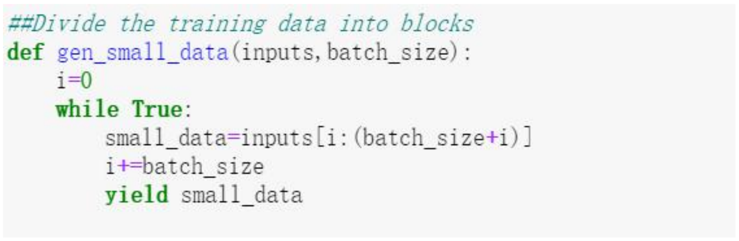
	Learning rate	Number of training	The accuracy of training set	The accuracy of test set
tf.train.AdamOptimizer	0.01	2	≈1.0	0.9
tf.train.AdamOptimizer	0.01	5	≈1.0	≈1.0
tf.train.AdamOptimizer	0.001	2	0.95455	0.85
tf.train.AdamOptimizer	0.001	5	0.86	0.86

From the table above, when the learning rate is the same, the increase of training times can increase the accuracy rate. When the training times are the same, the higher the learning rate, the higher the accuracy.
Next, we need to define the optimizer. Here we choose the Cross-entropy loss function as the loss function of the optimizer. Another advantage of using Cross-entropy as loss function is that sigmoid function can avoid the problem of lower learning rate of mean square error loss function when gradient decreases.



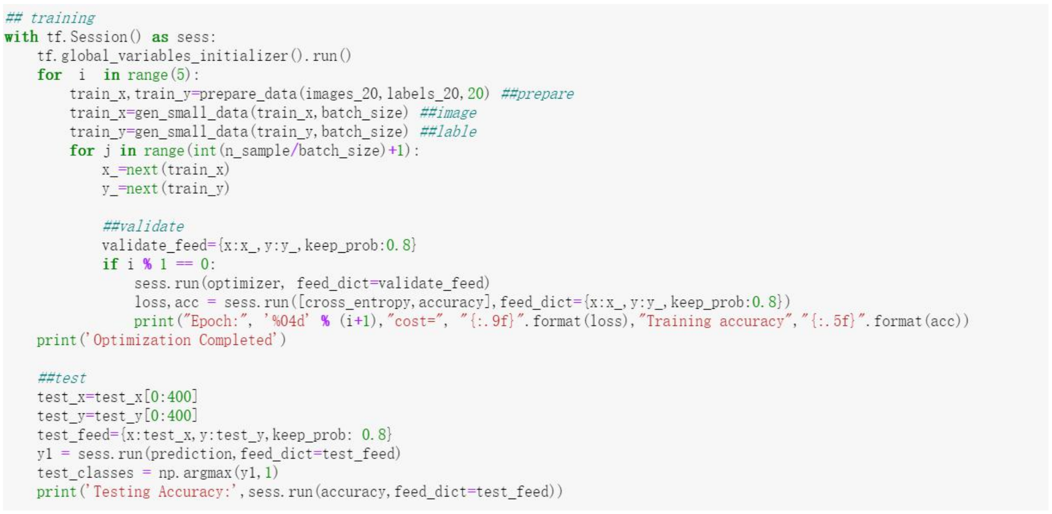
```
## Prediction
prediction=conv_net(x,Weights,bias,keep_prob) ##Convolutional neural network
cross_entropy=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=prediction,labels=y)) ##cross-entropy cost function
optimizer=tf.train.AdamOptimizer(0.0009).minimize(cross_entropy) ##Optimizer and learning rate


##Accuracy
correct_pred=tf.equal(tf.argmax(prediction,1),tf.argmax(y,1))
accuracy=tf.reduce_mean(tf.cast(correct_pred,tf.float32))
```

⑥ Checking the accuracy of the model on the test set
First, we define a function to block the training data.



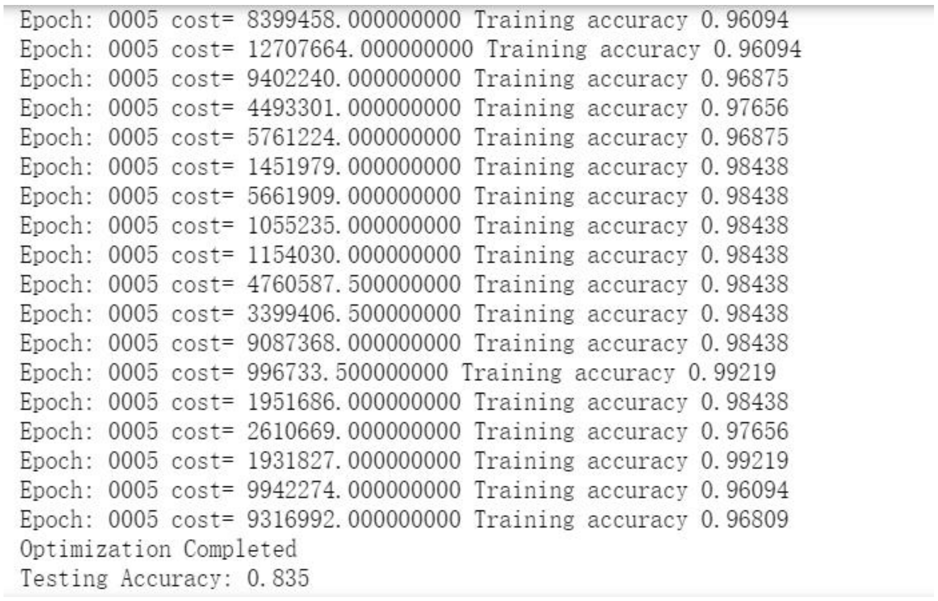
```
##Divide the training data into blocks
def gen_small_data(inputs,batch_size):
    i=0
    while True:
        small_data=inputs[i:(batch_size+i)]
        i+=batch_size
        yield small_data
```

Next, we train the data and evaluate the training model.



```
## training
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for  i  in range(5):
        train_x,train_y=prepare_data(images_20,labels_20,20) ##prepare
        train_x=gen_small_data(train_x,batch_size) ##image
        train_y=gen_small_data(train_y,batch_size) ##lable
        for j in range(int(n_sample/batch_size)+1):
            x_=next(train_x)
            y_=next(train_y)

            ##validate
            validate_feed={x:x_,y:y_,keep_prob:0.8}
            if i % 1 == 0:
                sess.run(optimizer, feed_dict=validate_feed)
                loss,acc = sess.run([cross_entropy,accuracy],feed_dict={x:x_,y:y_,keep_prob:0.8})
                print("Epoch:", '%04d' % (i+1),"cost=", "{:.9f}".format(loss),"Training accuracy","{:.5f}".format(acc))
    print('Optimization Completed')

    ##test
    test_x=test_x[0:400]
    test_y=test_y[0:400]
    test_feed={x:test_x,y:test_y,keep_prob: 0.8}
    y1 = sess.run(prediction,feed_dict=test_feed)
    test_classes = np.argmax(y1,1)
    print('Testing Accuracy:',sess.run(accuracy,feed_dict=test_feed))
```

The following is the training set accuracy and test set accuracy of 20 kinds of fruits after 5 times of training with Adam optimizer. The accuracy of the test set is about 0.83, and the model is effective.

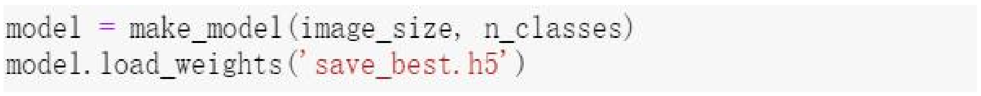


```
Epoch: 0005 cost= 8399458.000000000 Training accuracy 0.96094
Epoch: 0005 cost= 12707664.000000000 Training accuracy 0.96094
Epoch: 0005 cost= 9402240.000000000 Training accuracy 0.96875
Epoch: 0005 cost= 4493301.000000000 Training accuracy 0.97656
Epoch: 0005 cost= 5761224.000000000 Training accuracy 0.96875
Epoch: 0005 cost= 1451979.000000000 Training accuracy 0.98438
Epoch: 0005 cost= 5661909.000000000 Training accuracy 0.98438
Epoch: 0005 cost= 1055235.000000000 Training accuracy 0.98438
Epoch: 0005 cost= 1154030.000000000 Training accuracy 0.98438
Epoch: 0005 cost= 4760587.500000000 Training accuracy 0.98438
Epoch: 0005 cost= 3399406.500000000 Training accuracy 0.98438
Epoch: 0005 cost= 9087368.000000000 Training accuracy 0.98438
Epoch: 0005 cost= 996733.500000000 Training accuracy 0.99219
Epoch: 0005 cost= 1951686.000000000 Training accuracy 0.98438
Epoch: 0005 cost= 2610669.000000000 Training accuracy 0.97656
Epoch: 0005 cost= 1931827.000000000 Training accuracy 0.99219
Epoch: 0005 cost= 9942274.000000000 Training accuracy 0.96094
Epoch: 0005 cost= 9316992.000000000 Training accuracy 0.96809
Optimization Completed
Testing Accuracy: 0.835
```

⑦ Using OpenCV to visualize the model

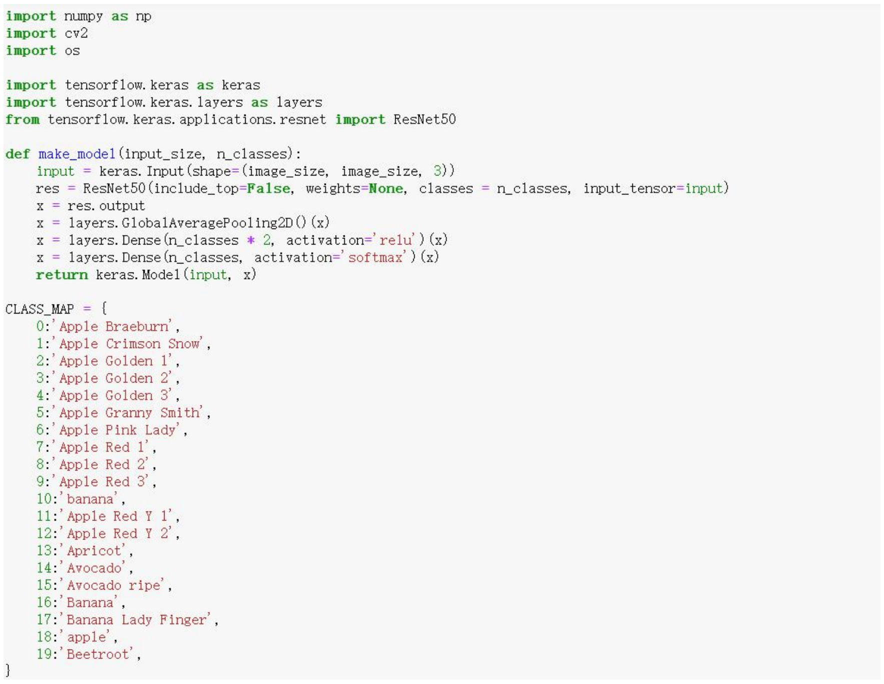
In the last part of this experiment, we use opencv for visualization.

First of all, we read in the model we trained before.



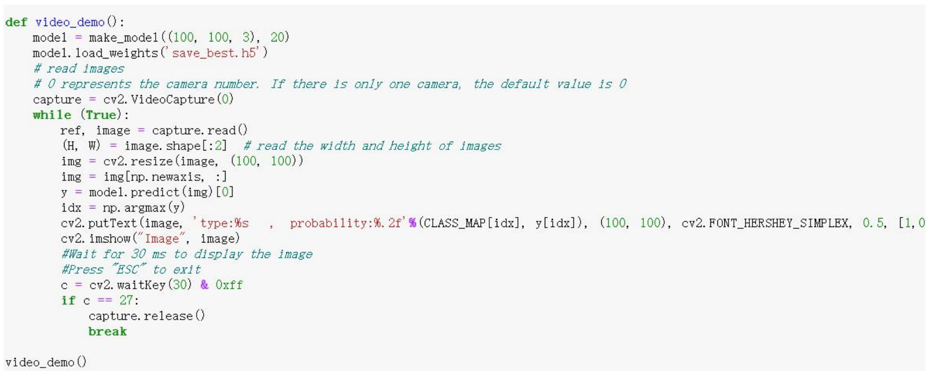
```
model = make_model(image_size, n_classes)
model.load_weights('save_best.h5')
```

Import the required library functions, and code for 20 kinds of fruits.



```
import numpy as np
import cv2
import os

import tensorflow.keras as keras
import tensorflow.keras.layers as layers
from tensorflow.keras.applications.resnet import ResNet50

def make_model(input_size, n_classes):
    input = keras.Input(shape=(image_size, image_size, 3))
    res = ResNet50(include_top=False, weights=None, classes = n_classes, input_tensor=input)
    x = res.output
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(n_classes * 2, activation='relu')(x)
    x = layers.Dense(n_classes, activation='softmax')(x)
    return keras.Model(input, x)

CLASS_MAP = {
    0:'Apple Braeburn',
    1:'Apple Crimson Snow',
    2:'Apple Golden 1',
    3:'Apple Golden 2',
    4:'Apple Golden 3',
    5:'Apple Granny Smith',
    6:'Apple Pink Lady',
    7:'Apple Red 1',
    8:'Apple Red 2',
    9:'Apple Red 3',
    10:'banana',
    11:'Apple Red Y 1',
    12:'Apple Red Y 2',
    13:'Apricot',
    14:'Avocado',
    15:'Avocado ripe',
    16:'Banana',
    17:'Banana Lady Finger',
    18:'apple',
    19:'Beetroot',
}
```

Define functions for visualization.



```python
def video_demo():
    model = make_model((100, 100, 3), 20)
    model.load_weights('save_best.h5')
    # read images
    # 0 represents the camera number. If there is only one camera, the default value is 0
    capture = cv2.VideoCapture(0)
    while (True):
        ref, image = capture.read()
        (H, W) = image.shape[:2]  # read the width and height of images
        img = cv2.resize(image, (100, 100))
        img = img[np.newaxis, :]
        y = model.predict(img)[0]
        idx = np.argmax(y)
        cv2.putText(image, 'type:%s   ,  probability:%.2f'%(CLASS_MAP[idx], y[idx]), (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5, [1,0
        cv2.imshow("Image", image)
        #Wait for 30 ms to display the image
        #Press "ESC" to exit
        c = cv2.waitKey(30) & 0xff
        if c == 27:
            capture.release()
            break

video_demo()
```

Discussion

The experiment completed the classification and detection of 20 kinds of fruits, and the specific display part can also be viewed in the appendix. We found that although the fruit species could be detected at last, the accuracy still fluctuated. In the later experiments, we will use other neural network models to train, and compare the accuracy. In addition, due to the capacity of experimental equipment, only 20 kinds of fruits were classified and tested in this experiment. If we can consider deploying the model to the front end, the sample number of the model will be expanded, and more kinds of fruits will be identified and detected by using larger equipment.

Appendix

Data: https://www.kaggle.com/moltean/fruits
Code: https://github.com/ucfnjqi/casa0018Assessment