计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目： Homework 3		学号：201900130024
日期：2021.10.30	班级： 数据 19	姓名： 刘士渤
Email：liuburger@qq.com		

实验目的：

完成 CNN step by step、CNN app 和 ResNet

实验软件和硬件环境：

VScode JupyterNoteBook

联想拯救者 Y7000p

实验原理和方法：

CNN

实验步骤：（不要求罗列完整源代码）

1. 补全 Convolution model-Step by Step.ipynb：

   卷积计算需要给原矩阵周围补 0（只在高和宽维度补 0）：

```
### START CODE HERE ### (≈ 1 line)
X_pad = np.pad(X,((0,0),(pad,pad),(pad,pad),(0,0)))
### END CODE HERE ###
```

   对于和卷积核相同大小的一小片，输出值是权重矩阵乘它，求和后再加上 bias

```
# Element-wise product between a_slice and W.
s = W*a_slice_prev
# Sum over all entries of the volume s  Add bias.
Z = np.sum(s)+b
```

conv_forward:

获取一些属性值、计算输出的维度、初始化输出矩阵、得到 pad

输出维度=（原矩阵维度-卷积核边长）/步长

```
# Retrieve dimensions from A_prev's shape (≈1 line
(m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape
# Retrieve dimensions from W's shape (≈1 line)
(f, f, n_C_prev, n_C) = W.shape
# Retrieve information from "hparameters" (≈2 line
stride = hparameters['stride']
pad = hparameters['pad']
# Compute the dimensions of the CONV output volume
n_H = int((n_H_prev-f+2*pad)/stride)+1
n_W = int((n_W_prev-f+2*pad)/stride)+1
# Initialize the output volume Z with zeros. (≈1 l
Z = np.zeros((m,n_H,n_W,n_C))
# Create A_prev_pad by padding A_prev
A_prev_pad = zero_pad(A_prev,pad)
```

生成 slice、利用之前写好的 conv_single_step 计算

```
for i in range(m):                               # loop over the batch of training examples
    a_prev_pad = A_prev_pad[i]                   # Select ith training example's padded act
    for h in range(n_H):                         # loop over vertical axis of the output vol
        for w in range(n_W):                     # loop over horizontal axis of the output v
            for c in range(n_C):                 # loop over channels (= #filters) of the ou
                #f是slice的边长
                # Find the corners of the current "slice" (≈4 lines)
                vert_start = h*stride
                vert_end = h*stride+f
                horiz_start = w*stride
                horiz_end = w*stride+f
                # Use the corners to define the (3D) slice of a_prev_pad (See Hint abo
                a_slice_prev = a_prev_pad[vert_start:vert_end,horiz_start:horiz_end,:]
                # Convolve the (3D) slice with the correct filter W and bias b, to get
                Z[i, h, w, c] = conv_single_step(a_slice_prev,W[:,:,:,c],b[0,0,0,c])
```

pool_forward:
根据模式的不同选择用 np.max 或 np.mean

```
for i in range(m):                          # loop over the training examples
    for h in range(n_H):                     # loop on the vertical axis of the outpu
        for w in range(n_W):                 # loop on the horizontal axis of the out
            for c in range (n_C):            # loop over the channels of the output v
                # Find the corners of the current "slice" (≈4 lines)
                vert_start = h*stride
                vert_end = vert_start+f
                horiz_start = w*stride
                horiz_end = horiz_start+f
                # Use the corners to define the current slice on the ith training exa
                a_prev_slice = A_prev[i,vert_start:vert_end,horiz_start:horiz_end,c]
                # Compute the pooling operation on the slice. Use an if statment to d
                if mode == "max":
                    A[i, h, w, c] = np.max(a_prev_slice)
                elif mode == "average":
                    A[i, h, w, c] = np.mean(a_prev_slice)
```

conv_backward:
获取一些属性值，初始化 dA_prev、dW、db，给矩阵加 pad:

```
# Retrieve information from "cache"
(A_prev, W, b, hparameters) = cache
# Retrieve dimensions from A_prev's shape
(m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape
# Retrieve dimensions from W's shape
(f, f, n_C_prev, n_C) = W.shape
# Retrieve information from "hparameters"
stride = hparameters['stride']
pad = hparameters['pad']
# Retrieve dimensions from dZ's shape
(m, n_H, n_W, n_C) = dZ.shape
# Initialize dA_prev, dW, db with the correct shapes
dA_prev = np.zeros(A_prev.shape)
dW = np.zeros(W.shape)
db = np.zeros(b.shape)
# Pad A_prev and dA_prev
A_prev_pad = zero_pad(A_prev,pad)
dA_prev_pad = zero_pad(dA_prev,pad)
```

由公式

$$dA+=\sum_{h=0}^{n_H}\sum_{w=0}^{n_W} W_c \times dZ_{hw}$$

$$dW_c+=\sum_{h=0}^{n_H}\sum_{w=0}^{n_W} a_{slice} \times dZ_{hw}$$

$$db=\sum_{h}\sum_{w} dZ_{hw}$$

更新 dA、dW、db 的值：

```
for i in range(m): # Loop over the training examples
    # select ith training example from A_prev_pad and dA_prev_pad
    a_prev_pad = A_prev_pad[i]
    da_prev_pad = dA_prev_pad[i]
    for h in range(n_H):                       # loop over vertical axis of the output volume
        for w in range(n_W):                   # loop over horizontal axis of the output volume
            for c in range(n_C):               # loop over the channels of the output volume
                # Find the corners of the current "slice"
                vert_start = h*stride
                vert_end = vert_start+f
                horiz_start = w*stride
                horiz_end = horiz_start+f
                # Use the corners to define the slice from a_prev_pad
                a_slice = a_prev_pad[vert_start:vert_end,horiz_start:horiz_end,:]
                # Update gradients for the window and the filter's parameters using the code formulas
                da_prev_pad[vert_start:vert_end, horiz_start:horiz_end, :] += W[:,:,:,c]*dZ[i,h,w,c]
                dW[:,:,:,c] += a_slice*dZ[i,h,w,c]
                db[:,:,:,c] += dZ[i,h,w,c]
    # Set the ith training example's dA_prev to the unpaded da_prev_pad (Hint: use X[pad:-pad, pad:-p
    dA_prev[i, :, :, :] = da_prev_pad[pad:-pad,pad:-pad,:]
```

create_mask_from_window：

所有不是 max 的都被 mask 掉：

```
mask = (np.max(x)==x)
```

distribute_value：

矩阵有多少个元素就系数就是多少分之一：

```
(n_H, n_W) = shape
# Compute the value to dist
average = dz/(n_H*n_W)
# Create a matrix where eve
a = average*np.ones(shape)
```

pool_backward：

获取一些属性值，初始化 dA_prev：

```
# Retrieve information from cache (≈1 line)
(A_prev, hparameters) = cache
# Retrieve hyperparameters from "hparameters" (
stride = hparameters['stride']
f = hparameters['f']
# Retrieve dimensions from A_prev's shape and d
m, n_H_prev, n_W_prev, n_C_prev = A_prev.shape
m, n_H, n_W, n_C = dA.shape
# Initialize dA_prev with zeros (≈1 line)
dA_prev = np.zeros(A_prev.shape)
```

如果模式是 max，用之前写好的 mask 保留最大值；

如果模式是 average，用之前写好的 distribute_value 求平均：

```
for i in range(m): # loop over the training examples
    # select training example from A_prev (≈1 line)
    a_prev = A_prev[i]
    for h in range(n_H):                    # loop on the vertical axis
        for w in range(n_W):                # loop on the horizontal axis
            for c in range(n_C):            # loop over the channels (depth)
                # Find the corners of the current "slice" (≈4 lines)
                vert_start = h*stride
                vert_end = vert_start+f
                horiz_start = w*stride
                horiz_end = horiz_start+f
                # Compute the backward propagation in both modes.
                if mode == "max":
                    # Use the corners and "c" to define the current slice from a_prev (≈1 line)
                    a_prev_slice = a_prev[vert_start:vert_end,horiz_start:horiz_end,c]
                    # Create the mask from a_prev_slice (≈1 line)
                    mask = create_mask_from_window(a_prev_slice)
                    # Set dA_prev to be dA_prev + (the mask multiplied by the correct entry of dA) (≈1 line)
                    dA_prev[i, vert_start: vert_end, horiz_start: horiz_end, c] += mask*dA[i, h, w, c]
                elif mode == "average":
                    # Get the value a from dA (≈1 line)
                    da = dA[i, h, w, c]
                    # Define the shape of the filter as fxf (≈1 line)
                    shape = [f,f]
                    # Distribute it to get the correct slice of dA_prev. i.e. Add the distributed value of da
                    dA_prev[i, vert_start: vert_end, horiz_start: horiz_end, c] += distribute_value(da,shape)
```

2. 补全 Convolution model-Application.ipynb

create_placeholders:

tensorflow 的 placeholder 可以满足 batchsize 为 None:

```
### START CODE HERE ### (≈2 lines)
X = tf.placeholder('float',[None, n_H0, n_W0, n_C0])
Y = tf.placeholder('float',[None, n_y])
### END CODE HERE ###
```

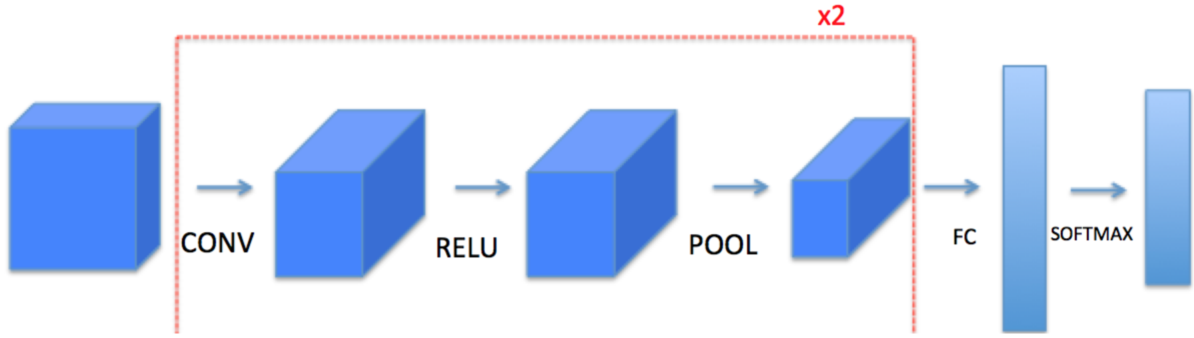
initialize_parameters:

用 seed=0 对 W1 和 W2 进行初始化:

```
### START CODE HERE ### (approx. 2 lines of code)
W1 = tf.get_variable('W1',[4, 4, 3, 8],initializer=tf.contrib.layers.xavier_initializer(seed = 0))
W2 = tf.get_variable('W2',[2, 2, 8, 16],initializer=tf.contrib.layers.xavier_initializer(seed = 0))
### END CODE HERE ###
```

forward_propagation:

两遍的卷积→Relu→最大值池化，然后变平化、全连接:




```
# CONV2D: stride of 1, padding 'SAME'
Z1 = tf.nn.conv2d(X,W1,strides=[1,1,1,1],padding='SAME')
# RELU
A1 = tf.nn.relu(Z1)
# MAXPOOL: window 8x8, sride 8, padding 'SAME'
P1 = tf.nn.max_pool(A1,ksize=[1,8,8,1],strides=[1,8,8,1],padding='SAME')
# CONV2D: filters W2, stride 1, padding 'SAME'
Z2 = tf.nn.conv2d(P1,W2,strides=[1,1,1,1],padding='SAME')
# RELU
A2 = tf.nn.relu(Z2)
# MAXPOOL: window 4x4, stride 4, padding 'SAME'
P2 = tf.nn.max_pool(A2,ksize=[1,4,4,1],strides=[1,4,4,1],padding='SAME')
# FLATTEN
P2 = tf.contrib.layers.flatten(P2)
# FULLY-CONNECTED without non-linear activation function (do not call so
# 6 neurons in output layer. Hint: one of the arguments should be "activ
Z3 = tf.contrib.layers.fully_connected(P2,6,activation_fn=None)
```

compute_cost:

计算 softmax 熵损失，并对其求均值:

```
### START CODE HERE ### (1 line of code)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=Z3,labels=Y))
### END CODE HERE ###
```

model:

调用之前写好的函数，使用 AdamOptimizer 进行优化:

```
### START CODE HERE ### (1 line)
X, Y = create_placeholders(n_H0,n_W0,n_C0,n_y)
### END CODE HERE ###
# Initialize parameters
### START CODE HERE ### (1 line)
parameters = initialize_parameters()
### END CODE HERE ###
# Forward propagation: Build the forward propagation in the tenso
### START CODE HERE ### (1 line)
Z3 = forward_propagation(X,parameters)
### END CODE HERE ###
# Cost function: Add cost function to tensorflow graph
### START CODE HERE ### (1 line)
cost = compute_cost(Z3,Y)
### END CODE HERE ###
# Backpropagation: Define the tensorflow optimizer. Use an AdamOp
### START CODE HERE ### (1 line)
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
### END CODE HERE ###
```

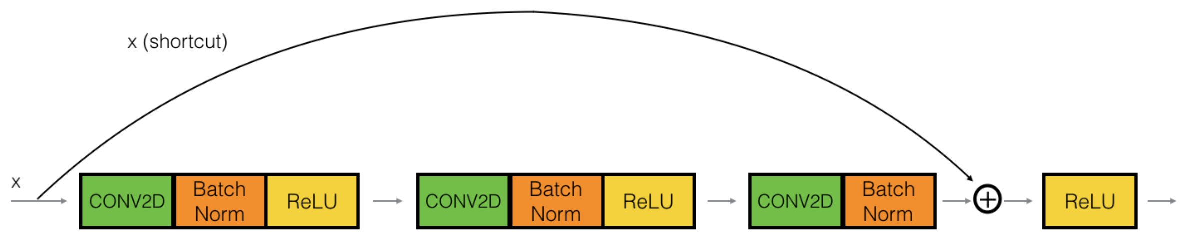
调用 sess 执行 optimizer 和 cost:

```
### START CODE HERE ### (1 line)
_ , temp_cost = sess.run([optimizer,cost],feed_dict={X:minibatch_X,Y:minibatch_Y})
### END CODE HERE ###
```

3. 补全 Residual Networks.ipynb

identity_block：

3 层的 main path，第二层是 conv、batchNorm 和 ReLU；第三层是 conv 和 batchNorm；最后将 shortcut 与 X 加到一起：

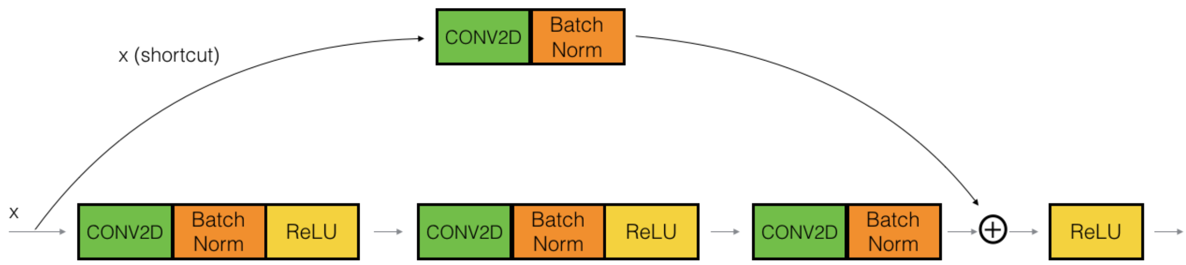


种子都是 0，第二层 padding 是 same，第三层 padding 是 valid：

```
# Second component of main path (≈3 lines)
X = Conv2D(F2,(f,f),strides=(1,1),padding='same',name=conv_name_base+'2b',kernel_initializer=glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
X = Activation('relu')(X)
# Third component of main path (≈2 lines)
X = Conv2D(F3,(1,1),strides=(1,1),padding='valid',name=conv_name_base+'2c',kernel_initializer=glorot_uniform(seed=0))(X)
X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)
# Final step: Add shortcut value to main path, and pass it through a RELU activation (≈2 lines)
X = Add()([X_shortcut,X])
X = Activation('relu')(X)
```

convolutional_block：

总体上与 identity_block 相同，但对 shortcut 做了 conv 和 BatchNorm：

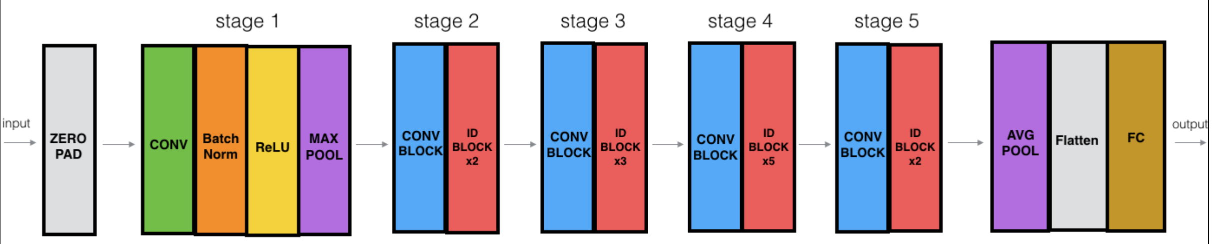


shortcut 和第三层一样都使用 F3 作为 filter：

```
##### SHORTCUT PATH #### (≈2 lines)
X_shortcut = Conv2D(F3,(1,1),strides=(s,s),padding='valid',name=conv_name_base+'1',
                    kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
X_shortcut = BatchNormalization(axis = 3, name = bn_name_base + '1')(X_shortcut)
```

ResNet50：

第 2 到 5 个 stage 都是 conv block+ identity_block：



stage3 用到 3 个 idblock 分别是 b、c、d；

stage4 用到 5 个 idblock 分别是 b、c、d、e、f；

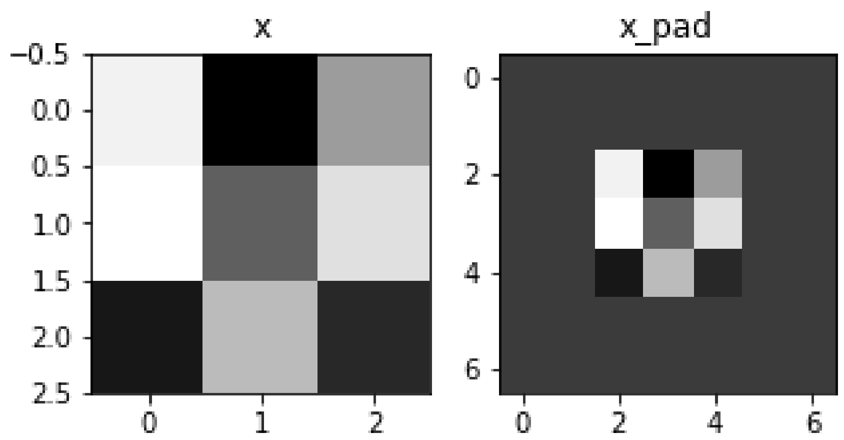
stage5 用到 2 个 idblock 分别是 b、c：

```
# Stage 3 (≈4 lines)
X = convolutional_block(X, f = 3, filters = [128, 128, 512], stage = 3, block='a', s = 2)
X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')
# Stage 4 (≈6 lines)
X = convolutional_block(X, f = 3, filters = [256, 256, 1024], stage = 4, block='a', s = 2)
X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')
# Stage 5 (≈3 lines)
X = convolutional_block(X, f = 3, filters = [512, 512, 2048], stage = 5, block='a', s = 2)
X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')
# AVGPOOL (≈1 line). Use "X = AveragePooling2D(...)(X)"
X = AveragePooling2D(pool_size=(2,2),name='avg_pool')(X)
```

结论分析与体会：

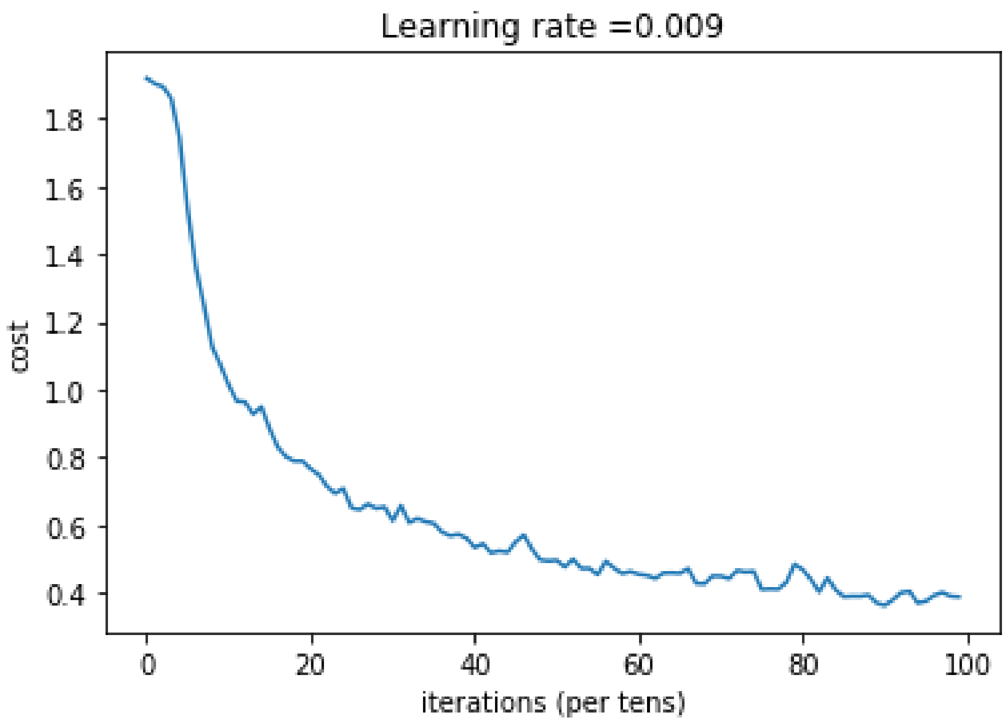
1. Convolution model-Step by Step：

   补 0 之前和之后的矩阵对比：

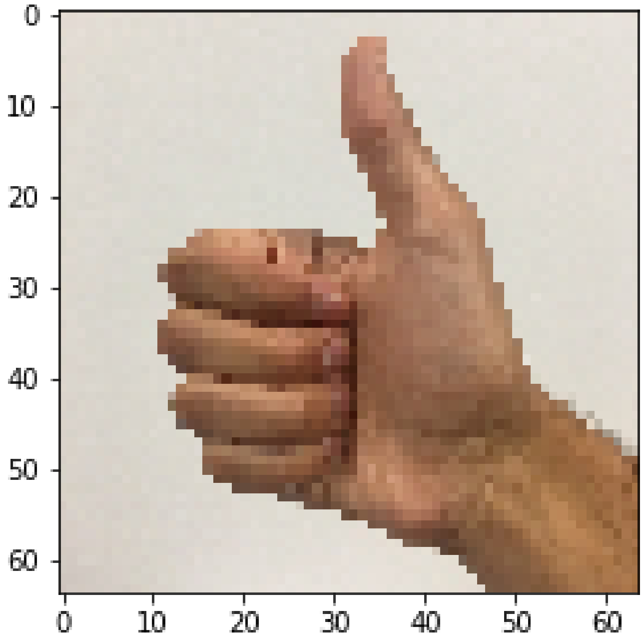


2. Convolution model-Application：

   在学习率为 0.009 的 tensor 模型中，cost 随迭代次数增加而下降，在 1000 次左右稳定到 0.4：



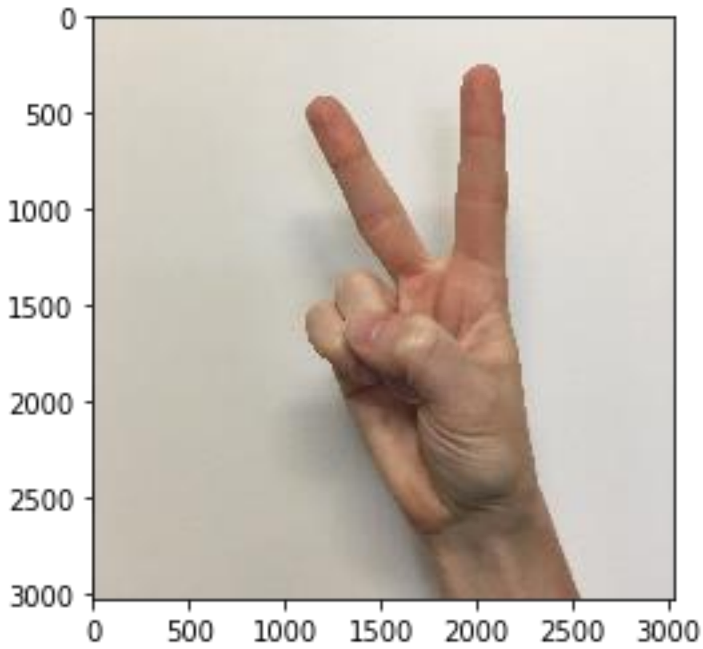
测试准确率为 0.87，预测准确率为 0.73：



3. Residual Networks：

训练出的 model 对图像进行预测，向量显示图片的手势是 0：

```
Input image shape: (1, 64, 64, 3)
class prediction vector [p(0), p(1), p(2), p(3), p(4), p(5)] =
[[1. 0. 0. 0. 0. 0.]]
```



有一张很长的图片展示了整个 ResNet 的步骤，这里放不下了。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

1. 初次运行 ResNet50 的时候，报这样的错误：

```
D:\Anaconda3\envs\cnn\lib\site-packages\keras\engine\topology.py in build_map_of_graph(tensor, finished_nodes, node
x)
   1675            """
   1676            if not layer or node_index is None or tensor_index is None:
-> 1677                layer, node_index, tensor_index = tensor._keras_history
   1678            node = layer.inbound_nodes[node_index]
   1679

AttributeError: 'Tensor' object has no attribute '_keras_history'
```

以为是配的环境有问题，但其实包的版本都是对的；
百度之后，知道了原因出在之前 Relu 部分的代码：

```
X += X_shortcut
X = Activation('relu')(X)
```

由于 X 和 X_short 是 tensor 类型，这样就会默认调用 tensorflow 的加和函数，导致最后的结果还是 tensor 类型；keras 的数据类型和 tensorflow 的数据类型不同，两者不能够混用。

使用 keras 自带的求和层 Add() 可以解决：

```
X = Add()([X_shortcut,X])
X = Activation('relu')(X)
```

其实代码的开头导入的包是一个提示：

```
import numpy as np
from keras.layers import Input, Add, Dense, Activation,
from keras.models import Model, load_model
```

2. ResNet 的预测结果是 0，而与输入图片是 2，反复检查自己的代码没有问题，怀疑是题目的代码有一些问题。

```
class prediction vector [p(0), p(1), p(2), p(3), p(4), p(5)] =
[[1. 0. 0. 0. 0. 0.]]
```