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：Convolution model Residual Networks		学号：201900130015
日期：2021	班级： 智能班	姓名：李德锋
Email：ldf2878945468@163.com		

实验目的：

根据公式补全代码，并测试

实验软件和硬件环境：

Intel(R) Core(TM) i7-8550U CPU

华为云

实验原理和方法：

根据公式补全代码，并测试

实验步骤：（不要求罗列完整源代码）

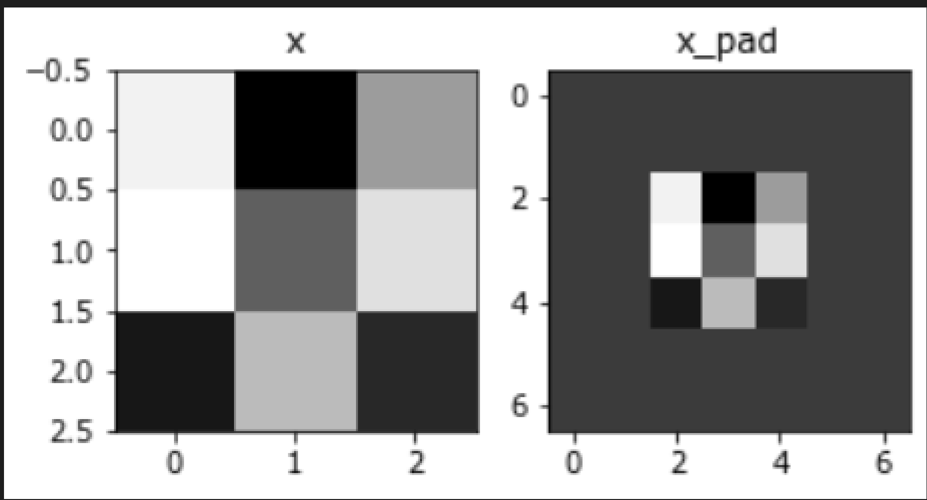
Convolutional Neural Networks: Step by Step

导入所需的包

零填充 使用 np.pad

```
X_pad = np.pad(X,((0,0),(pad,pad),(pad,pad),(0,0)),'constant',constant_values = (0,0))
```

```
x.shape = (4, 3, 3, 2)
x_pad.shape = (4, 7, 7, 2)
x[1,1] = [[ 0.90085595 -0.68372786]
 [-0.12289023 -0.93576943]
 [-0.26788808  0.53035547]]
x_pad[1,1] = [[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
<matplotlib.image.AxesImage at 0x24e3b94b130>
```



单步卷积

实现一个卷积步骤，将过滤器应用到输入的单个位置。左侧矩阵中的每个值对应于单个像素值，我们通过将其值逐元素乘以原始矩阵，然后将它们相加，将 3x3 过滤器与图像进行卷积

```
s = np.multiply(a_slice_prev,W)
# Sum over all entries of the volume s
Z = np.sum(s)+b
```

```
Z = [[[-6.99908945]]]
```

卷积神经网络 - 前向传递

在输入激活 A_prev 上对滤波器 W 进行卷积

```
# 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 lines)
stride = hparameters["stride"]
pad = hparameters["pad"]

# Compute the dimensions of the CONV output volume using
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 line)
Z = np.zeros((m, n_H, n_W, n_C))
```

```
A_prev_pad = zero_pad(A_prev,pad)

for i in range(m):                               # loop over the batch of training exam
    a_prev_pad = A_prev_pad[i]                              # Select ith training examp
    for h in range(n_H):                           # loop over vertical axis of the out
        for w in range(n_W):                       # loop over horizontal axis of the c
            for c in range(n_C):                   # loop over channels (= #filters) of

                # 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 (3D) slice of a_prev_pad (See Hint abov
                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[...,c])
```

```
Z's mean = 0.048995203528855794
Z[3,2,1] = [-0.61490741 -6.7439236  -2.55153897  1.75698377  3.56208902  0.53036437
  5.18531798  8.75898442]
cache_conv[0][1][2][3] = [-0.20075807  0.18656139  0.41005165]
```

池化层

池化（POOL）层减少了输入的高度和宽度。它有助于减少计算，并有助于使特征检测器对其在输入中的位置更加不变

前向池

$$n_H = \lfloor \frac{n_{H_{prev}} - f}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f}{stride} \rfloor + 1$$

$$n_C = n_{C_{prev}}$$

```python
for i in range(m):                         # loop over the trai
    for h in range(n_H):                    # loop on the vert
        for w in range(n_W):                # loop on the hori
            for c in range (n_C):           # loop over the ch

                # Find the corners of the current "slice" (≈4 l
                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 o
                a_prev_slice = A_prev[i, vert_start:vert_end, h

                # Compute the pooling operation on the slice. U
                if mode == "max":
                    A[i, h, w, c] = np.max(a_prev_slice)
                elif mode == "average":
                    A[i, h, w, c] = np.average(a_prev_slice)
```

```
mode = max
A = [[[[1.74481176 0.86540763 1.13376944]]]


 [[[1.13162939 1.51981682 2.18557541]]]]

mode = average
A = [[[[ 0.02105773 -0.20328806 -0.40389855]]]


 [[[-0.22154621  0.51716526  0.48155844]]]]
```

卷积神经网络中的反向传播

计算 dA, dW

```
# 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((m, n_H_prev, n_W_prev, n_C_prev))
dW = np.zeros((f, f, n_C_prev, n_C))
db = np.zeros((1, 1, 1, n_C))

# Pad A_prev and dA_prev
A_prev_pad = zero_pad(A_prev, pad)
dA_prev_pad = zero_pad(dA_prev, pad)
```

```python
for i in range(m):                            # loop over the trainin
    
    # select ith training example from A_prev_pad and dA_prev_pa
    a_prev_pad = A_prev_pad[i,:]
    da_prev_pad = dA_prev_pad[i,:]
    
    for h in range(n_H):                      # loop over vertical
        for w in range(n_W):                  # loop over horizonta
            for c in range(n_C):              # loop over the chann
                
                # 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_pre
                a_slice = a_prev_pad[vert_start:vert_end, horiz_
                
                
                
                # Update gradients for the window and the filter
                da_prev_pad[vert_start:vert_end, horiz_start:hor
                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_p
    dA_prev[i, :, :, :] = da_prev_pad[pad:-pad, pad:-pad, :]
```

```
dA_mean = 1.4524377775388075
dW_mean = 1.7269914583139097
db_mean = 7.839232564616838
```

池化层——反向传播

构建一个名为的辅助函数 create_mask_from_window()

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

```
x =  [[ 1.62434536 -0.61175641 -0.52817175]
 [-1.07296862  0.86540763 -2.3015387 ]]
mask =  [[ True False False]
 [False False False]]
```

平均池化 - 反向传播

在最大池化中，对于每个输入窗口，对输出的所有“影响”都来自单个输入值——最大值。在平均池化中，输入窗口的每个元素对输出都有相同的影响

```
### START CODE HERE ###
# Retrieve dimensions from shape (≈1 line)
(n_H, n_W) = shape

# Compute the value to distribute on the matrix (≈1 line)
average = n_H * n_W

# Create a matrix where every entry is the "average" value (≈1 line)
a = np.ones(shape) * dz / average
### END CODE HERE ###
```

```
distributed value = [[0.5 0.5]
 [0.5 0.5]]
```

组合起来：向后池

```
# Retrieve information from cache (≈1 line)
(A_prev, hparameters) = cache

# Retrieve hyperparameters from "hparameters" (≈2 lines)
stride = hparameters['stride']
f = hparameters['f']

# Retrieve dimensions from A_prev's shape and dA's shape (≈2 lines)
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_like(A_prev)
```

```
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 (
                    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 e
                    dA_prev[i, vert_start: vert_end, horiz_start: horiz_end, c] += np.m

                elif mode == "average":

                    # Get the value a from dA (≈1 line)
                    da = np.mean(dA[i, vert_start:vert_end, horiz_start:horiz_end, 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 c
```

```
mode = max
mean of dA =  0.145713902729l8056
dA_prev[1,1] =  [[ 0.          0.        ]
 [10.11330283 -0.49726956]
 [ 0.          0.        ]]

mode = average
mean of dA =  0.14571390272918056
dA_prev[1,1] =  [[ 2.59843096 -0.27835778]
 [ 7.96018612 -1.95394424]
 [ 5.36175516 -1.67558646]]
```

Convolutional Neural Networks: Application

创建占位符

```
X = tf.placeholder(tf.float32,[None, n_H0, n_W0, n_C0] )
Y = tf.placeholder(tf.float32,[None, n_y])
```

初始化参数

```
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))
```

```
W1 = [ 0.00131723  0.1417614  -0.04434952  0.09197326  0.14984085 -0.03514394
 -0.06847463  0.05245192]
W2 = [-0.08566415  0.17750949  0.11974221  0.16773748 -0.0830943  -0.08058
 -0.00577033 -0.14643836  0.24162132 -0.05857408 -0.19055021  0.1345228
 -0.22779644 -0.1601823  -0.16117483 -0.10286498]
```

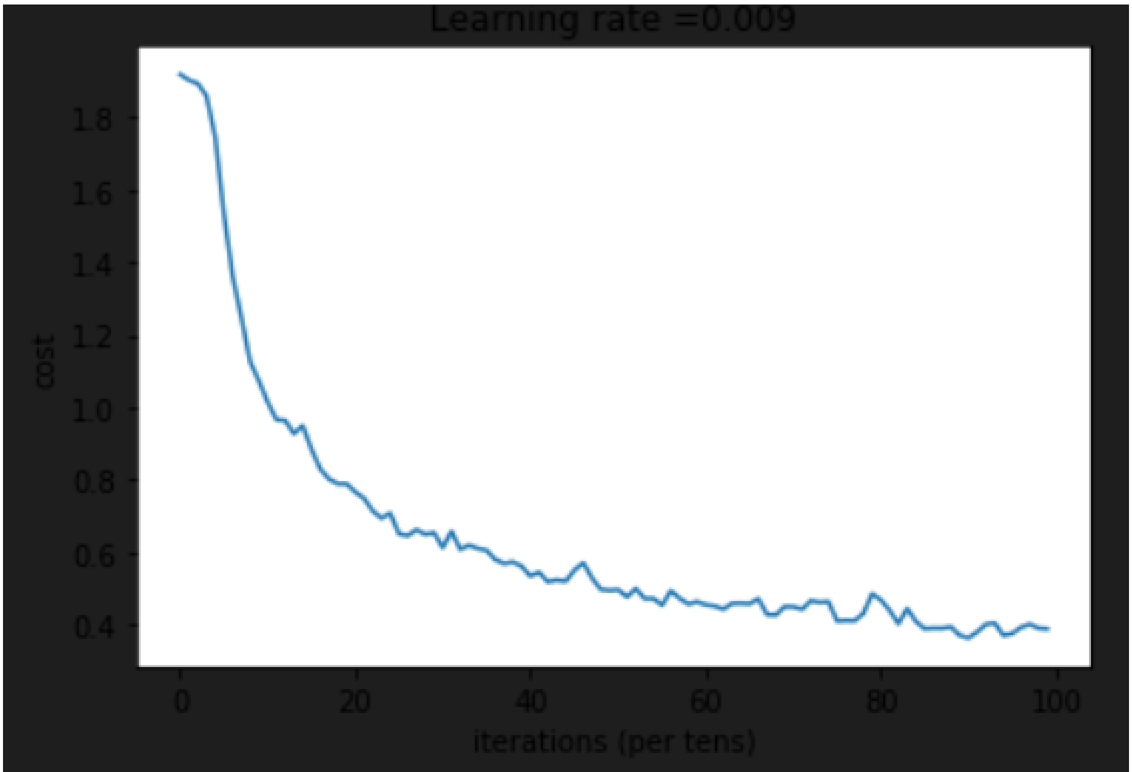
前向传播

```
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 (not not call softmax).
# 6 neurons in output layer. Hint: one of the arguments should be "activation_fn=None"
Z3 = tf.contrib.layers.fully_connected(P2,num_outputs=6,activation_fn=None)
```

计算损失

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = Z3, labels = Y))
```

构建模型，创建优化器




```
Tensor("Mean_1:0", shape=(), dtype=float32)
Train Accuracy: 0.86851853
Test Accuracy: 0.73333335
```

构建 ResNet 模型 零填充用(3,3)填充填充输入，阶段 3，4，5 具体参数给出

Residual Networks
构建残差网络，允许梯度直接反向传播到较早的层
实现 ResNet 标识块

```
# Second component of main path (≈3 lines)
X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1,1), padding = 'same', name = conv_name_base + '2b',
X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
X = Activation('relu')(X)

# Third component of main path (≈2 lines)
X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name = conv_name_base + '2c',
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, X_shortcut])
X = Activation('relu')(X)
```

卷积块：当输入和输出维度不匹配，CONV2D 层用于将输入大小调整 为不同的维度

```
# Second component of main path (≈3 lines)
X = Conv2D(F2, (f, f), strides = (1,1), name = conv_name_base + '2b',padding='same', kernel_initializ
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), name = conv_name_base + '2c',padding='valid', kernel_initiali
X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

##### SHORTCUT PATH #### (≈2 lines)
X_shortcut = Conv2D(filters=F3, kernel_size=(1, 1), strides=(s, s), padding='valid', name=conv_name_b
X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)

# Final step: Add shortcut value to main path, and pass it through a RELU activation (≈2 lines)
X = Add()([X, X_shortcut])
X = Activation('relu')(X)
```

构建 ResNet 模型 零填充用(3,3)填充填充输入，阶段 3，4，5 具体参数给出

```
# 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')
```

训练结果

```
Epoch 1/2
1080/1080 [==============================] - 81s 75ms/step - loss: 3.2086 - acc: 0.2454
Epoch 2/2
1080/1080 [==============================] - 82s 76ms/step - loss: 2.2404 - acc: 0.3296
```

```
120/120 [==============================] - 3s 24ms/step
Loss = 2.3448499043782554
Test Accuracy = 0.16666666666666666
```

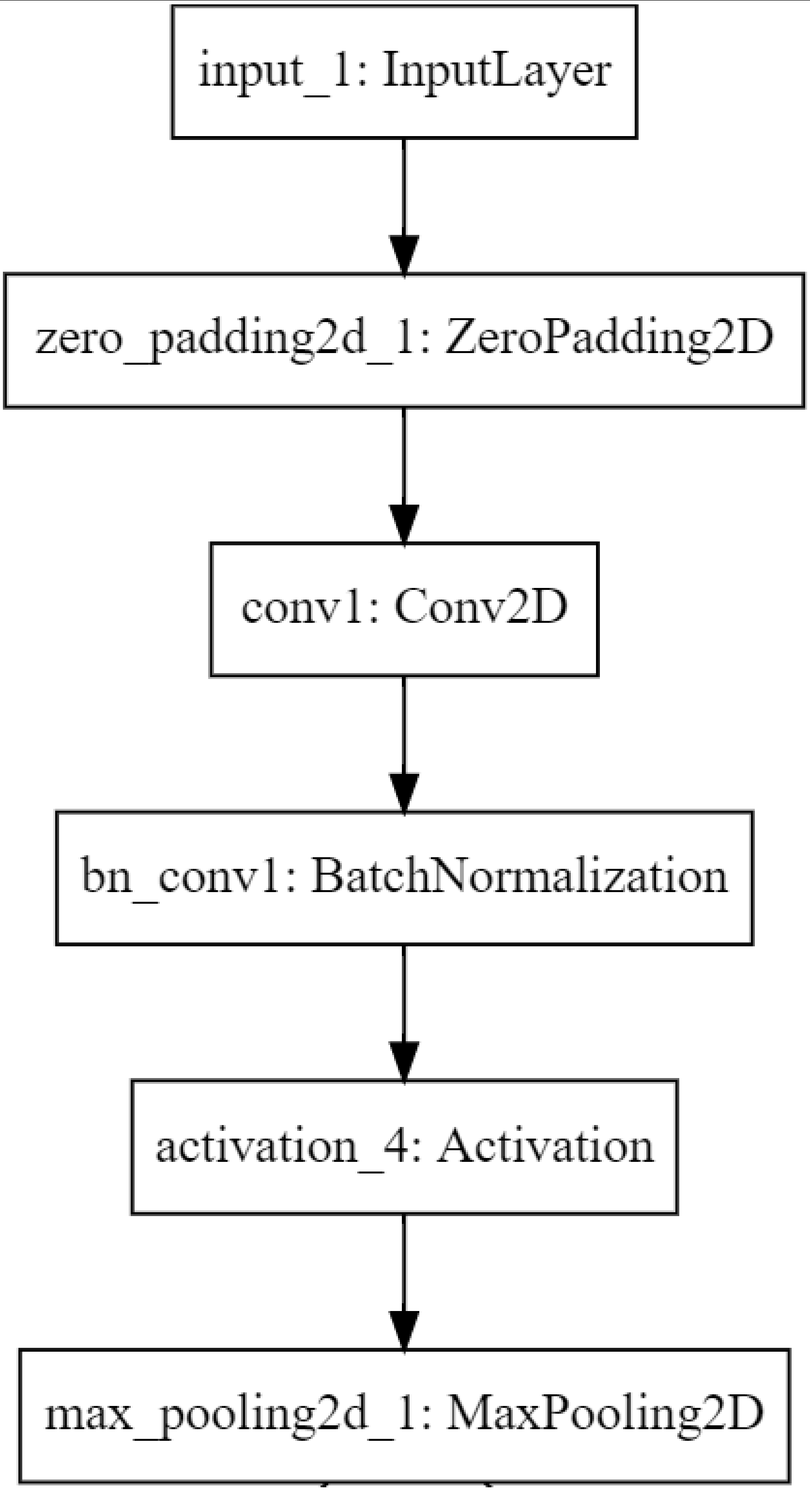
在自己图像进行测试

输出模型摘要

```
model.summary()
```

```
Output exceeds the size limit. Open the full output data in a text editor
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 64, 64, 3)    0
__________________________________________________________________________________________________
zero_padding2d_1 (ZeroPadding2D (None, 70, 70, 3)    0           input_1[0][0]
__________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 32, 32, 64)   9472        zero_padding2d_1[0][0]
__________________________________________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 32, 32, 64)   256         conv1[0][0]
__________________________________________________________________________________________________
```

可视化模型



结论分析与体会：

Convolutional Neural Networks: Step by Step

学会了实现的卷积函数：零填充、卷积窗口、向前卷积、向后卷积；池化功能

零填充允许使用 CONV 层而不必缩小体积的高度和宽度，帮助我们在图像的边界保留更多信息

池化（POOL）层减少了输入的高度和宽度。它有助于减少计算，并有助于使特征检测器

对其在输入中的位置更加不变 Convolutional Neural Networks: Application 步骤：创建占位符，初始化参数，前向传播，计算损失，创建优化器 Residual Networks 层数多的网络的可以表示更加复杂的功能，还可以学习许多不同抽象级别的特征，但是训练它们的一个巨大障碍是梯度下降慢 残差网络，可以训练更深的网络，跳跃连接有助于解决梯度消失问题
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： 运行结果与提示的答案不同 查询后发现时 tensorflow 的版本问题，修改版本后答案一致