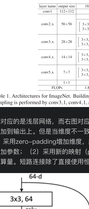
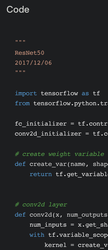
ResNet



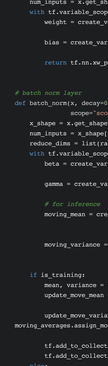
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

左图对应的是浅层网络，而右图对应的是深层网络。对于短路连接，当输入和输出维度一致时，可以直接将输入加到输出上。但是当维度不一致时（对应的是维度增加一倍），这就不能直接相加。有两种策略：（1）采用zero-padding增加维度，此时一般要先做一个downsamp，可以采用stride=2的pooling，这样不会增加参数；（2）采用新的映射（projection shortcut），一般采用1x1的卷积，这样会增加参数，也会增加计算量。短路连接除了直接使用恒等映射，当然都可以采用projection shortcut。



Structure



Code

```
---
ResNet50
2017/12/06
---


import tensorflow as tf
from tensorflow.python.training import moving_averages


fc_initializer = tf.contrib.layers.xavier_initializer
conv2d_initializer = tf.contrib.layers.xavier_initializer_conv2d

# create weight variable
def create_var(name, shape, initializer, trainable=True):
    return tf.get_variable(name, shape=shape, dtype=tf.float32,
                           initializer=initializer, trainable=trainable)


# conv2d layer
def conv2d(x, num_outputs, kernel_size, stride=1, scope="conv2d"):
    num_inputs = x.get_shape()[-1]
    with tf.variable_scope(scope):
        kernel = create_var("kernel", [kernel_size, kernel_size,
                                       num_inputs, num_outputs],
                            conv2d_initializer())
        return tf.nn.conv2d(x, kernel, strides=[1, stride, stride, 1],
                            padding="SAME")


# fully connected layer
def fc(x, num_outputs, scope="fc"):
    num_inputs = x.get_shape()[-1]
    with tf.variable_scope(scope):
        weight = create_var("weight", [num_inputs, num_outputs],
                            fc_initializer())
        bias = create_var("bias", [num_outputs,],
                          tf.zeros_initializer())
        return tf.nn.xw_plus_b(x, weight, bias)


# batch norm layer
def batch_norm(x, decay=0.999, epsilon=1e-03, is_training=True,
               scope="scope"):
    x_shape = x.get_shape()
    num_inputs = x_shape[-1]
    reduce_dims = list(range(len(x_shape) - 1))
    with tf.variable_scope(scope):
        beta = create_var("beta", [num_inputs,],
                          initializer=tf.zeros_initializer())
        gamma = create_var("gamma", [num_inputs,],
                           initializer=tf.ones_initializer())
        # for inference
        moving_mean = create_var("moving_mean", [num_inputs,],
                                 initializer=tf.zeros_initializer(),
                                 trainable=False)
        moving_variance = create_var("moving_variance", [num_inputs],
                                     initializer=tf.ones_initializer(),
                                     trainable=False)
        if is_training:
            mean, variance = tf.nn.moments(x, axes=reduce_dims)
            update_move_mean = moving_averages.assign_moving_average(moving_mean,
                                                                     mean, decay=decay)
            update_move_variance = moving_averages.assign_moving_average(moving_variance,
                                                                         variance, decay=decay)
            tf.add_to_collection(tf.GraphKeys.UPDATE_OPS, update_move_mean)
            tf.add_to_collection(tf.GraphKeys.UPDATE_OPS, update_move_variance)
        else:
            mean, variance = moving_mean, moving_variance
        return tf.nn.batch_normalization(x, mean, variance, beta, gamma, epsilon)


# avg pool layer
def avg_pool(x, pool_size, scope):
    with tf.variable_scope(scope):
        return tf.nn.avg_pool(x, [1, pool_size, pool_size, 1],
                              strides=[1, pool_size, pool_size, 1], padding="VALID")

# max pool layer
def max_pool(x, pool_size, stride, scope):
    with tf.variable_scope(scope):
        return tf.nn.max_pool(x, [1, pool_size, pool_size, 1],
                              [1, stride, stride, 1], padding="SAME")


class ResNet50(object):
    def __init__(self, inputs, num_classes=1000, is_training=True,
                 scope="resnet50"):
        self.inputs =inputs
        self.is_training = is_training
        self.num_classes = num_classes

        with tf.variable_scope(scope):
            # construct the model
            net = conv2d(inputs, 64, 7, 2, scope="conv1") # -> [batch, 112, 112, 64]
            net = tf.nn.relu(batch_norm(net, is_training=self.is_training,
scope="bn1"))
            net = max_pool(net, 3, 2, scope="maxpool1")  # -> [batch, 56, 56, 64]
            net = self._block(net, 256, 3, init_stride=1,
is_training=self.is_training,
                              scope="block2")             # -> [batch, 56, 56, 256]
            net = self._block(net, 512, 4, is_training=self.is_training,
scope="block3")
                                                          # -> [batch, 28, 28, 512]
            net = self._block(net, 1024, 6, is_training=self.is_training,
scope="block4")
                                                          # -> [batch, 14, 14, 1024]
            net = self._block(net, 2048, 3, is_training=self.is_training,
scope="block5")
                                                          # -> [batch, 7, 7, 2048]
            net = avg_pool(net, 7, scope="avgpool5")      # -> [batch, 1, 1, 2048]
            net = tf.squeeze(net, [1, 2], name="SpatialSqueeze") # -> [batch, 2048]
            self.logits = fc(net, self.num_classes, "fc6")        # -> [batch, num_classes]
            self.predictions = tf.nn.softmax(self.logits)


    def _block(self, x, n_out, n, init_stride=2, is_training=True,
scope="block"):
        with tf.variable_scope(scope):
            h_out = n_out // 4
            out = self._bottleneck(x, h_out, n_out, stride=init_stride,
                                   is_training=is_training,
scope="bottleneck1")
            for i in range(1, n):
                out = self._bottleneck(out, h_out, n_out,
is_training=is_training,
                                       scope="bottleneck%s" % (i + 1))
            return out

    def _bottleneck(self, x, h_out, n_out, stride=None, is_training=True,
scope="bottleneck"):
        """ A residual bottleneck unit"""
        n_in = x.get_shape()[-1]
        if stride is None:
            stride = 1 if n_in == n_out else 2

        with tf.variable_scope(scope):
            h = conv2d(x, h_out, 1, stride=stride, scope="conv_1")
            h = batch_norm(h, is_training=is_training, scope="bn_1")
            h = tf.nn.relu(h)
            h = conv2d(h, h_out, 3, stride=1, scope="conv_2")
            h = batch_norm(h, is_training=is_training, scope="bn_2")
            h = tf.nn.relu(h)
            h = conv2d(h, n_out, 1, stride=1, scope="conv_3")
            h = batch_norm(h, is_training=is_training, scope="bn_3")

            if n_in != n_out:
                shortcut = conv2d(x, n_out, 1, stride=stride, scope="conv_4")
                shortcut = batch_norm(shortcut, is_training=is_training,
scope="bn_4")
            else:
                shortcut = x
            return tf.nn.relu(shortcut + h)

if __name__ == "__main__":
    x = tf.random_normal([32, 224, 224, 3])
    resnet50 = ResNet50(x)
    print(resnet50.logits)
```