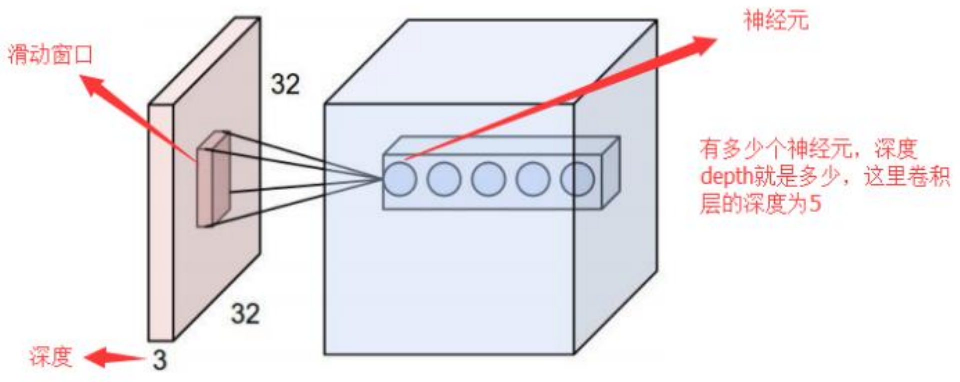
LeNet/Inception_v3
In Mnist Data

陈俊华 2017104087

CNN

- 结构:Input layer/Conv layer/Relu layer/Pooling layer/FC layer
- Input layer:
  - 对原始图像的预处理，如规范输入维度，去均值，归一化，PCA
- Conv layer:
  - 卷积操作



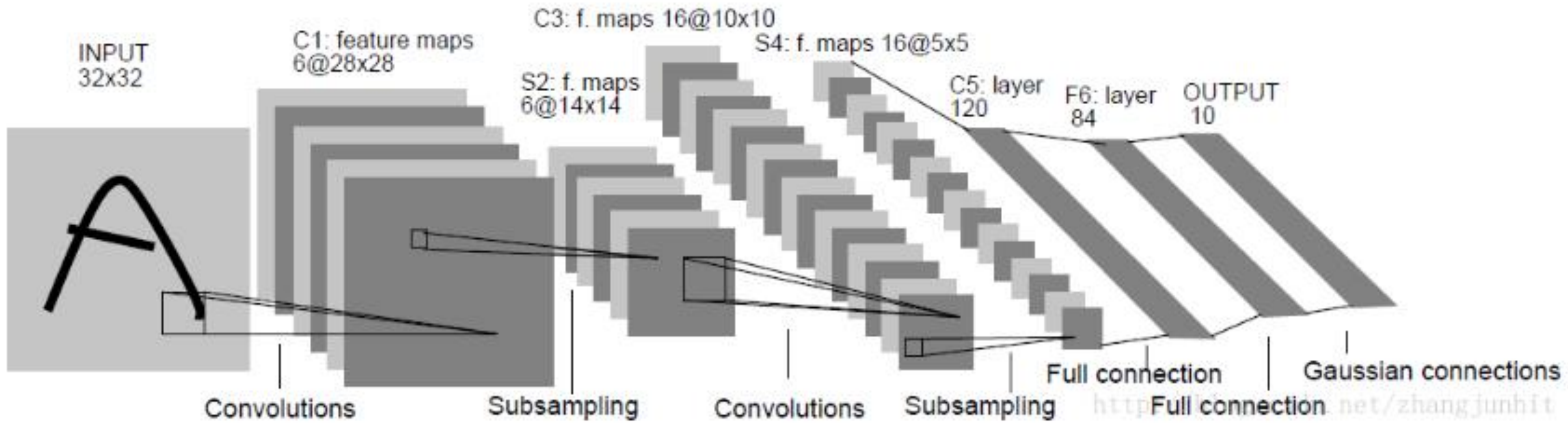
- Relu layer:
  - 对卷积层输出结果做非线性映射，激活函数为Relu

CNN

- Pooling layer：
  - 池化层夹在连续的卷积层中间， 用于压缩数据和参数的量，减小过拟合。简而言之，如果输入是图像的话，那么池化层的最主要作用就是压缩图像。Max Pooling,Average Pooling。
- FC layer:
  - 两层之间所有神经元都有权重连接，通常全连接层在卷积神经网络尾部。也就是跟传统的神经网络神经元的连接方式是一样的。

LeNET-5

- LeNet-5是LeCUN于1998年提出，是第一个成功应用于数字识别问题的CNN。LetNet-5模型总共6层，如下图所示：



LeNet-5

- 第一层卷积层:过滤器尺寸5x5，深度为6

```
# 第一层卷积层
with tf.variable_scope('layer1-conv1'):
    conv1_weights = tf.get_variable(
        "weight", [CONV1_SIZE, CONV1_SIZE, NUM_CHANNELS, CONV1_DEEP],
        initializer=tf.truncated_normal_initializer(stddev=0.1))
    conv1_biases = tf.get_variable("bias", [CONV1_DEEP], initializer=tf.constant_initializer(0.0))
    conv1 = tf.nn.conv2d(input_tensor, conv1_weights, strides=[1, 1, 1, 1], padding='SAME')
    relu1 = tf.nn.relu(tf.nn.bias_add(conv1, conv1_biases))
```

- 第二层池化层：过滤器尺寸2x2

```
# 第二层池化层
with tf.name_scope("layer2-pool1"):
    pool1 = tf.nn.max_pool(relu1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")
```

LeNet-5

- 第三层卷积层:过滤器尺寸5x5，深度为16

```
# 第三层卷积层
with tf.variable_scope("layer3-conv2"):
    conv2_weights = tf.get_variable(
        "weight", [CONV2_SIZE, CONV2_SIZE, CONV1_DEEP, CONV2_DEEP],
        initializer=tf.truncated_normal_initializer(stddev=0.1))
    conv2_biases = tf.get_variable("bias", [CONV2_DEEP], initializer=tf.constant_initializer(0.0))
    conv2 = tf.nn.conv2d(pool1, conv2_weights, strides=[1, 1, 1, 1], padding='SAME')
    relu2 = tf.nn.relu(tf.nn.bias_add(conv2, conv2_biases))
```

- 第四层池化层：过滤器尺寸2x2，步长为2

```
# 第四层池化层
with tf.name_scope("layer4-pool2"):
    pool2 = tf.nn.max_pool(relu2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    pool_shape = pool2.get_shape().as_list()
    nodes = pool_shape[1] * pool_shape[2] * pool_shape[3]
    reshaped = tf.reshape(pool2, [pool_shape[0], nodes])
```

LeNet-5

- 第五、六层全连接层：

```
# 第五层全连接层
with tf.variable_scope('layer5-fc1'):
    fc1_weights = tf.get_variable("weight", [nodes, FC_SIZE],
                                  initializer=tf.truncated_normal_initializer(stddev=0.1))
    if regularizer != None: tf.add_to_collection('losses', regularizer(fc1_weights))
    fc1_biases = tf.get_variable("bias", [FC_SIZE], initializer=tf.constant_initializer(0.1))

    fc1 = tf.nn.relu(tf.matmul(reshaped, fc1_weights) + fc1_biases)

# 第六层全连接层
with tf.variable_scope('layer6-fc2'):
    fc2_weights = tf.get_variable("weight", [FC_SIZE, NUM_LABELS],
                                  initializer=tf.truncated_normal_initializer(stddev=0.1))
    if regularizer != None: tf.add_to_collection('losses', regularizer(fc2_weights))
    fc2_biases = tf.get_variable("bias", [NUM_LABELS], initializer=tf.constant_initializer(0.1))
    logit = tf.matmul(fc1, fc2_weights) + fc2_biases

# 返回最终的前向传播输出
return logit
```

LeNet-5

- 数据集：MNIST手写数据集
- 实验结果：

```
Net
Extracting datasets/MNIST_data\t10k-images-idx3-ubyte.gz
Extracting datasets/MNIST_data\t10k-labels-idx1-ubyte.gz
input/x-input:0
2018-06-12 13:50:05.086034: I C:\tf_jenkins\home\workspace\rel-win\
After 1 training step(s), the accuracy is 0.11.
After 101 training step(s), the accuracy is 0.93.
After 201 training step(s), the accuracy is 0.92.
After 301 training step(s), the accuracy is 0.91.
After 401 training step(s), the accuracy is 0.89.
After 501 training step(s), the accuracy is 0.93.
After 601 training step(s), the accuracy is 0.96.
After 701 training step(s), the accuracy is 0.94.
After 801 training step(s), the accuracy is 0.96.
After 901 training step(s), the accuracy is 0.97.
After 1001 training step(s), the accuracy is 0.96.
After 1101 training step(s), the accuracy is 0.93.
After 1201 training step(s), the accuracy is 0.98.
```

Inception_v3

- Inceptionjiang将不同的卷积层通过并联的方式结合在一起。如图：

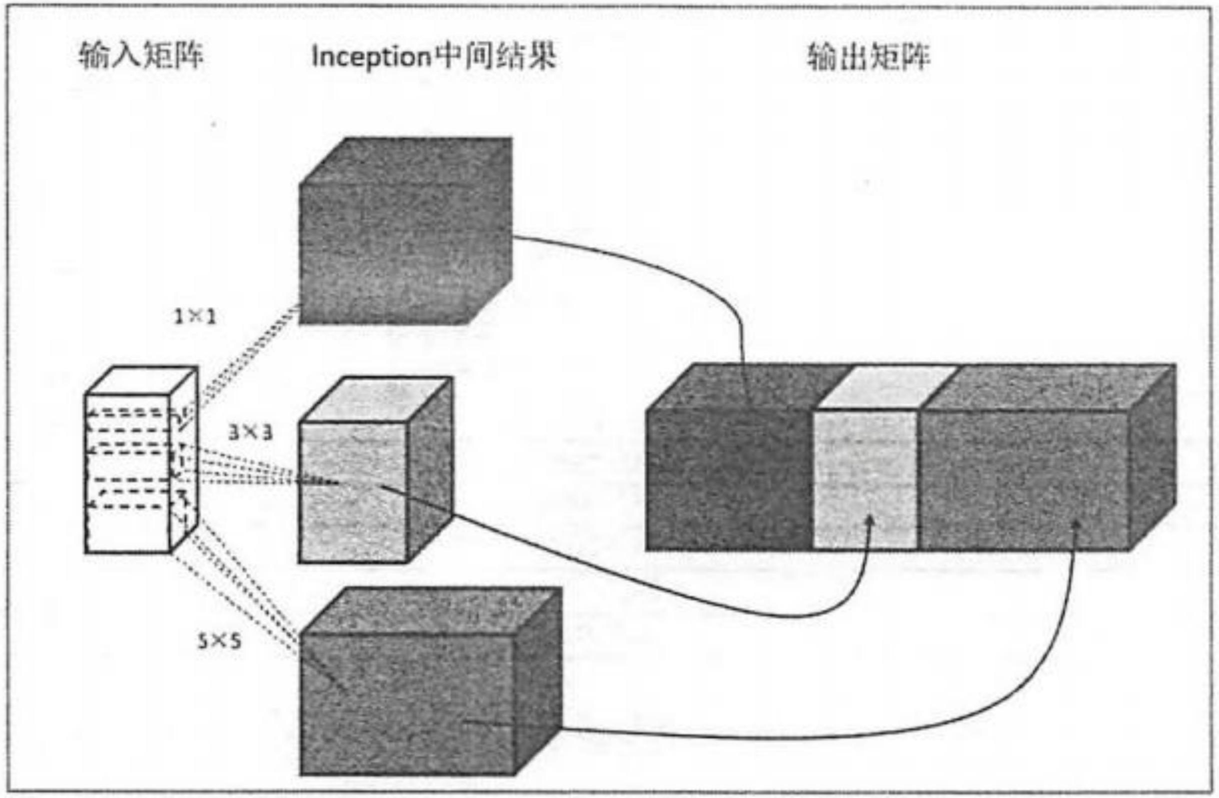


图 6-16 Inception 模块示意图。

Inception_v3

```python
with tf.variable_scope('Mixed_7c'):
    with tf.variable_scope('Branch_0'):
        branch_0 = slim.conv2d(net, 320, [1, 1], scope='Conv2d_0a_1x1')
    with tf.variable_scope('Branch_1'):
        branch_1 = slim.conv2d(net, 384, [1, 1], scope='Conv2d_0a_1x1')
        branch_1 = tf.concat([
            slim.conv2d(branch_1, 384, [1, 3], scope='Conv2d_0b_1x3'),
            slim.conv2d(branch_1, 384, [3, 1], scope='Conv2d_0c_3x1')
        ], 3)
    with tf.variable_scope('Branch_2'):
        branch_2 = slim.conv2d(net, 448, [1, 1], scope='Conv2d_0a_1x1')
        branch_2 = slim.conv2d(branch_2, 384, [3, 3], scope='Conv2d_0b_3x3')
        branch_2 = tf.concat([
            slim.conv2d(branch_2, 384, [1, 3], scope='Conv2d_0c_1x3'),
            slim.conv2d(branch_2, 384, [3, 1], scope='Conv2d_0d_3x1')
        ], 3)
    with tf.variable_scope('Branch_3'):
        branch_3 = slim.avg_pool2d(net, [3, 3], scope='AvgPool_0a_3x3')
        branch_3 = slim.conv2d(branch_3, 192, [1, 1], scope='Conv2d_0b_1x1')
    net = tf.concat([branch_0, branch_1, branch_2, branch_3], 3)
return net, end_points
```

Inception_v3

- 实验结果：

```
D:\python\python.exe "E:/Program Files (x86)/Pycharm/Project/Test/CNN Ne
2018-06-12 13:54:19.423582: I C:\tf_jenkins\home\workspace\rel-win\M\wi
Step 0: Validation accuracy on random sampled 100 examples = 42.0%
2018-06-12 13:54:25.527931: W C:\tf_jenkins\home\workspace\rel-win\M\wi
Step 100: Validation accuracy on random sampled 100 examples = 84.0%
Step 200: Validation accuracy on random sampled 100 examples = 84.0%
Step 300: Validation accuracy on random sampled 100 examples = 86.0%
Step 400: Validation accuracy on random sampled 100 examples = 89.0%
Step 500: Validation accuracy on random sampled 100 examples = 97.0%
Step 600: Validation accuracy on random sampled 100 examples = 90.0%
Step 700: Validation accuracy on random sampled 100 examples = 91.0%
Step 800: Validation accuracy on random sampled 100 examples = 92.0%
Step 900: Validation accuracy on random sampled 100 examples = 88.0%
Step 1000: Validation accuracy on random sampled 100 examples = 92.0%
```

```
Step 3900: Validation accuracy on random sampled 100 examples = 94.0%
Step 3999: Validation accuracy on random sampled 100 examples = 93.0%
Final test accuracy = 92.9%
```