TensorFlow人工智能引擎入门教程之十 最强网络 RSNN深度残差网络 平均准确率96-99%

在第六届 ImageNet 图像识别挑战赛上，微软研究院在多个类别的比赛中取得了第一名的成绩。比赛结果显示，微软的技术水平远远超越了 Google、Intel、高通、腾讯以及一众创业公司和科研实验室。

这个叫做「图像识别的深度残差学习」的获胜项目由微软研究员何恺明、张祥雨、任少卿和孙剑共同完成。根据微软博客显示，有关该成果的细节将会在后续的论文中详细介绍。

该技术的显著意义主要在于其复杂性。

http://arxiv.org/pdf/1512.03385v1.pdf

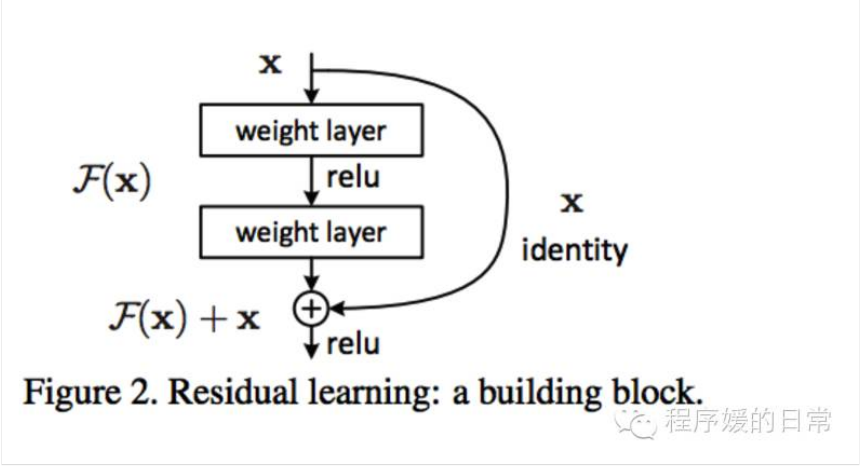

Figure 2. Residual learning: a building block.

因为传统的 多层网络 随着层数增多，导致残差 加大。所以为了防止这个问题，我们把多个网络看成一个单元，单元计算后将上次的产生的残差 记入 并记入下一次单元计算， 举个例子， 小明 拿出100块 买了一件1 元的 2 元的 6元的东西，但是 老板没有1块零钱，但是小明可能会继续买，所以 买了1 2 6 元 后当做10元 我买了3次 ，那么实际上相当于 每一次 老板还欠小明1块，总共3块，所以把这个3块 加入到下一次计算的里面呢，比如小明 下次买了个2元 5元的东西 那么实际上 就是3 2 5 记入下一次 网络，总之 我的理解就是 把每一次计算后 得到的残差 作为 作为一层网络 来替代，也就是说 把残差用网络替代， 就好像 我们 用wx+b 替代y 一样 ，实际的值 与 真实的值 有误差 所以 如果我们把这个误差 记入下一次wx+b来替代，最后是不是可以保证 中间每一层 wx+b 被抵押消除了 。大概是这样的，这是我的理解，网上也没有任何资料指出，个人看官方论文有感，如果有什么不对请指正。

看看官方samples的 关键代码。他使用3 3 3 的卷积核，三次卷积之后 产生的残差 记入 下一次 卷积，看net+conv 然后继续wx+b 参数了残差之后 继续把新产生的残差记入 下下次wx+b

```
# Loop through all res blocks
for block_i, block in enumerate(blocks):
    for repeat_i in range(block.num_repeats):

        name = 'block_%d/repeat_%d' % (block_i, repeat_i)
        conv = conv2d(net, block.bottleneck_size, k_h=1, k_w=1,
                      padding='VALID', stride_h=1, stride_w=1,
                      activation=activation,
                      name=name + '/conv_in')

        conv = conv2d(conv, block.bottleneck_size, k_h=3, k_w=3,
                      padding='SAME', stride_h=1, stride_w=1,
                      activation=activation,
                      name=name + '/conv_bottleneck')

        conv = conv2d(conv, block.num_filters, k_h=1, k_w=1,
                      padding='VALID', stride_h=1, stride_w=1,
                      activation=activation,
                      name=name + '/conv_out')

        net = conv + net
    try:
        # upscale to the next block size
        next_block = blocks[block_i + 1]
        net = conv2d(net, next_block.num_filters, k_h=1, k_w=1,
                     padding='SAME', stride_h=1, stride_w=1, bias=False,
                     name='block_%d/conv_upscale' % block_i)
    except IndexError:
        pass

# %%
```


2015 MSRE大赛第一名 准确率最高的深度学习网络，也是至今为止准确率最高的网络 幸运的是 在 好的训练集情况下 大概 结果 对大多数训练得到的准确率96-99.9之间 。rsnn152

152的太长了 ，这里贴出一个rsnn50 准确率 92-99

name: "ResNet-50" input: "data" input_dim: 1 input_dim: 3 input_dim: 224 input_dim: 224 layer { bottom: "data" top: "conv1" name: "conv1" type: "Convolution" convolution_param { num_output: 64 kernel_size: 7 pad: 3 stride: 2 } } layer { bottom: "conv1" top: "conv1" name: "bn_conv1" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "conv1" top: "conv1" name: "scale_conv1" type: "Scale" scale_param { bias_term: true } } layer { bottom: "conv1" top: "conv1" name: "conv1_relu" type: "ReLU" } layer { bottom: "conv1" top: "pool1" name: "pool1" type: "Pooling" pooling_param { kernel_size: 3 stride: 2 pool: MAX } } layer { bottom: "pool1" top: "res2a_branch1" name: "res2a_branch1" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2a_branch1" top: "res2a_branch1" name: "bn2a_branch1" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2a_branch1" top: "res2a_branch1" name: "scale2a_branch1" type: "Scale" scale_param { bias_term: true } } layer { bottom: "pool1" top: "res2a_branch2a" name: "res2a_branch2a" type: "Convolution" convolution_param { num_output: 64 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2a_branch2a" top: "res2a_branch2a" name: "bn2a_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2a_branch2a" top: "res2a_branch2a" name: "scale2a_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2a_branch2a" top: "res2a_branch2a" name: "res2a_branch2a_relu" type: "ReLU" } layer { bottom: "res2a_branch2a" top: "res2a_branch2b" name: "res2a_branch2b" type: "Convolution" convolution_param { num_output: 64 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res2a_branch2b" top: "res2a_branch2b" name: "bn2a_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2a_branch2b" top: "res2a_branch2b" name: "scale2a_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2a_branch2b" top: "res2a_branch2b" name: "res2a_branch2b_relu" type: "ReLU" } layer { bottom: "res2a_branch2b" top: "res2a_branch2c" name: "res2a_branch2c" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2a_branch2c" top: "res2a_branch2c" name: "bn2a_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2a_branch2c" top: "res2a_branch2c" name: "scale2a_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2a_branch1" bottom: "res2a_branch2c" top: "res2a" name: "res2a" type: "Eltwise" } layer { bottom: "res2a" top: "res2a" name: "res2a_relu" type: "ReLU" } layer { bottom: "res2a" top: "res2b_branch2a" name: "res2b_branch2a" type: "Convolution" convolution_param { num_output: 64 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2b_branch2a" top: "res2b_branch2a" name: "bn2b_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2b_branch2a" top: "res2b_branch2a" name: "scale2b_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2b_branch2a" top: "res2b_branch2a" name: "res2b_branch2a_relu" type: "ReLU" } layer { bottom: "res2b_branch2a" top: "res2b_branch2b" name: "res2b_branch2b" type: "Convolution" convolution_param { num_output: 64 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res2b_branch2b" top: "res2b_branch2b" name: "bn2b_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2b_branch2b" top: "res2b_branch2b" name: "scale2b_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2b_branch2b" top: "res2b_branch2b" name: "res2b_branch2b_relu" type: "ReLU" } layer { bottom: "res2b_branch2b" top: "res2b_branch2c" name: "res2b_branch2c" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2b_branch2c" top: "res2b_branch2c" name: "bn2b_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2b_branch2c" top: "res2b_branch2c" name: "scale2b_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2a" bottom: "res2b_branch2c" top: "res2b" name: "res2b" type: "Eltwise" } layer { bottom: "res2b" top: "res2b" name: "res2b_relu" type: "ReLU" } layer { bottom: "res2

b" top: "res2c_branch2a" name: "res2c_branch2a" type: "Convolution" convolution_param { num_output: 64 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2c_branch2a" top: "res2c_branch2a" name: "bn2c_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2c_branch2a" top: "res2c_branch2a" name: "scale2c_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2c_branch2a" top: "res2c_branch2a" name: "res2c_branch2a_relu" type: "ReLU" } layer { bottom: "res2c_branch2a" top: "res2c_branch2b" name: "res2c_branch2b" type: "Convolution" convolution_param { num_output: 64 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res2c_branch2b" top: "res2c_branch2b" name: "bn2c_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2c_branch2b" top: "res2c_branch2b" name: "scale2c_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2c_branch2b" top: "res2c_branch2b" name: "res2c_branch2b_relu" type: "ReLU" } layer { bottom: "res2c_branch2b" top: "res2c_branch2c" name: "res2c_branch2c" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res2c_branch2c" top: "res2c_branch2c" name: "bn2c_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res2c_branch2c" top: "res2c_branch2c" name: "scale2c_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2b" bottom: "res2c_branch2c" top: "res2c" name: "res2c" type: "Eltwise" } layer { bottom: "res2c" top: "res2c" name: "res2c_relu" type: "ReLU" } layer { bottom: "res2c" top: "res3a_branch1" name: "res3a_branch1" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 2 bias_term: false } } layer { bottom: "res3a_branch1" top: "res3a_branch1" name: "bn3a_branch1" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3a_branch1" top: "res3a_branch1" name: "scale3a_branch1" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res2c" top: "res3a_branch2a" name: "res3a_branch2a" type: "Convolution" convolution_param { num_output: 128 kernel_size: 1 pad: 0 stride: 2 bias_term: false } } layer { bottom: "res3a_branch2a" top: "res3a_branch2a" name: "bn3a_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3a_branch2a" top: "res3a_branch2a" name: "scale3a_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3a_branch2a" top: "res3a_branch2a" name: "res3a_branch2a_relu" type: "ReLU" } layer { bottom: "res3a_branch2a" top: "res3a_branch2b" name: "res3a_branch2b" type: "Convolution" convolution_param { num_output: 128 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res3a_branch2b" top: "res3a_branch2b" name: "bn3a_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3a_branch2b" top: "res3a_branch2b" name: "scale3a_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3a_branch2b" top: "res3a_branch2b" name: "res3a_branch2b_relu" type: "ReLU" } layer { bottom: "res3a_branch2b" top: "res3a_branch2c" name: "res3a_branch2c" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3a_branch2c" top: "res3a_branch2c" name: "bn3a_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3a_branch2c" top: "res3a_branch2c" name: "scale3a_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3a_branch1" bottom: "res3a_branch2c" top: "res3a" name: "res3a" type: "Eltwise" } layer { bottom: "res3a" top: "res3a" name: "res3a_relu" type: "ReLU" } layer { bottom: "res3a" top: "res3b_branch2a" name: "res3b_branch2a" type: "Convolution" convolution_param { num_output: 128 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3b_branch2a" top: "res3b_branch2a" name: "bn3b_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3b_branch2a" top: "res3b_branch2a" name: "scale3b_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3b_branch2a" top: "res3b_branch2a" name: "res3b_branch2a_relu" type: "ReLU" } layer { bottom: "res3b_branch2a" top: "res3b_branch2b" name: "res3b_branch2b" type: "Convolution" convolution_param { num_output: 128 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res3b_branch2b" top: "res3b_branch2b" name: "bn3b_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3b_branch2b" top: "res3b_branch2b" name: "scale3b_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3b_branch2b" top: "res3b_branch2b" name: "res3b_branch2b_relu" type: "ReLU" } layer { bottom: "res3b_branch2b" top: "res3b_branch2c" name: "res3b_branch2c" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3b_branch2c" top: "res3b_branch2c" name: "bn3b_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3b_branch2c" top: "res3b_branch2c" name: "scale3b_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3a" bottom: "res3b_branch2c" top: "res3b" name: "res3b" type: "Eltwise" } layer { bottom: "res3b" top: "res3b" name: "res3b_relu" type: "ReLU" } layer { bottom: "res3b" top: "res3c_branch2a" name: "res3c_branch2a" type: "Convolution" convolution_param { num_output: 128 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3c_branch2a" top: "res3c_branch2a" name: "bn3c_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3c_branch2a" top: "res3c_branch2a" name: "scale3c_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3c_branch2a" top: "res3c_branch2a" name: "res3c_branch2a_relu" type: "ReLU" } layer { bottom: "res3c_branch2a" top: "res3c_branch2b" name: "res3c_branch2b" type: "Convolution" convolution_param { num_output: 128 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res3c_branch2b" top: "res3c_branch2b" name: "bn3c_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3c_branch2b" top: "res3c_branch2b" name: "scale3c_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3c_branch2b" top: "res3c_branch2b" name: "res3c_branch2b_relu" type: "ReLU" } layer { bottom: "res3c_branch2b" top: "res3c_branch2c" name: "res3c_branch2c" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3c_branch2c" top: "res3c_branch2c" name: "bn3c_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3c_branch2c" top: "res3c_branch2c" name: "scale3c_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3b" bottom: "res3c_branch2c" top: "res3c" name: "res3c" type: "Eltwise" } layer { bottom: "res3c" top: "res3c" name: "res3c_relu" type: "ReLU" } layer { bottom: "res3c" top: "res3d_branch2a" name: "res3d_branch2a" type: "Convolution" convolution_param { num_output: 128 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3d_branch2a" top: "res3d_branch2a" name: "bn3d_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3d_branch2a" top: "res3d_branch2a" name: "scale3d_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3d_branch2a" top: "res3d_branch2a" name: "res3d_branch2a_relu" type: "ReLU" } layer { bottom: "res3d_branch2a" top: "res3d_branch2b" name: "res3d_branch2b" type: "Convolution" convolution_param { num_output: 128 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res3d_branch2b" top: "res3d_branch2b" name: "bn3d_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3d_branch2b" top: "res3d_branch2b" name: "scale3d_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3d_branch2b" top: "res3d_branch2b" name: "res3d_branch2b_relu" type: "ReLU" } layer { bottom: "res3d_branch2b" top: "res3d_branch2c" name: "res3d_branch2c" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res3d_branch2c" top: "res3d_branch2c" name: "bn3d_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res3d_branch2c" top: "res3d_branch2c" name: "sc

ale3d_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3c" bottom: "res3d_branch2c" top: "res3d" name: "res3d" type: "Eltwise" } layer { bottom: "res3d" top: "res3d" name: "res3d_relu" type: "ReLU" } layer { bottom: "res3d" top: "res4a_branch1" name: "res4a_branch1" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 stride: 2 bias_term: false } } layer { bottom: "res4a_branch1" top: "res4a_branch1" name: "bn4a_branch1" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4a_branch1" top: "res4a_branch1" name: "scale4a_branch1" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res3d" top: "res4a_branch2a" name: "res4a_branch2a" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 2 bias_term: false } } layer { bottom: "res4a_branch2a" top: "res4a_branch2a" name: "bn4a_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4a_branch2a" top: "res4a_branch2a" name: "scale4a_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4a_branch2a" top: "res4a_branch2a" name: "res4a_branch2a_relu" type: "ReLU" } layer { bottom: "res4a_branch2a" top: "res4a_branch2b" name: "res4a_branch2b" type: "Convolution" convolution_param { num_output: 256 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res4a_branch2b" top: "res4a_branch2b" name: "bn4a_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4a_branch2b" top: "res4a_branch2b" name: "scale4a_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4a_branch2b" top: "res4a_branch2b" name: "res4a_branch2b_relu" type: "ReLU" } layer { bottom: "res4a_branch2b" top: "res4a_branch2c" name: "res4a_branch2c" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4a_branch2c" top: "res4a_branch2c" name: "bn4a_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4a_branch2c" top: "res4a_branch2c" name: "scale4a_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4a_branch1" bottom: "res4a_branch2c" top: "res4a" name: "res4a" type: "Eltwise" } layer { bottom: "res4a" top: "res4a" name: "res4a_relu" type: "ReLU" } layer { bottom: "res4a" top: "res4b_branch2a" name: "res4b_branch2a" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4b_branch2a" top: "res4b_branch2a" name: "bn4b_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4b_branch2a" top: "res4b_branch2a" name: "scale4b_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4b_branch2a" top: "res4b_branch2a" name: "res4b_branch2a_relu" type: "ReLU" } layer { bottom: "res4b_branch2a" top: "res4b_branch2b" name: "res4b_branch2b" type: "Convolution" convolution_param { num_output: 256 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res4b_branch2b" top: "res4b_branch2b" name: "bn4b_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4b_branch2b" top: "res4b_branch2b" name: "scale4b_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4b_branch2b" top: "res4b_branch2b" name: "res4b_branch2b_relu" type: "ReLU" } layer { bottom: "res4b_branch2b" top: "res4b_branch2c" name: "res4b_branch2c" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4b_branch2c" top: "res4b_branch2c" name: "bn4b_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4b_branch2c" top: "res4b_branch2c" name: "scale4b_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4a" bottom: "res4b_branch2c" top: "res4b" name: "res4b" type: "Eltwise" } layer { bottom: "res4b" top: "res4b" name: "res4b_relu" type: "ReLU" } layer { bottom: "res4b" top: "res4c_branch2a" name: "res4c_branch2a" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4c_branch2a" top: "res4c_branch2a" name: "bn4c_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4c_branch2a" top: "res4c_branch2a" name: "scale4c_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4c_branch2a" top: "res4c_branch2a" name: "res4c_branch2a_relu" type: "ReLU" } layer { bottom: "res4c_branch2a" top: "res4c_branch2b" name: "res4c_branch2b" type: "Convolution" convolution_param { num_output: 256 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res4c_branch2b" top: "res4c_branch2b" name: "bn4c_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4c_branch2b" top: "res4c_branch2b" name: "scale4c_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4c_branch2b" top: "res4c_branch2b" name: "res4c_branch2b_relu" type: "ReLU" } layer { bottom: "res4c_branch2b" top: "res4c_branch2c" name: "res4c_branch2c" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4c_branch2c" top: "res4c_branch2c" name: "bn4c_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4c_branch2c" top: "res4c_branch2c" name: "scale4c_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4b" bottom: "res4c_branch2c" top: "res4c" name: "res4c" type: "Eltwise" } layer { bottom: "res4c" top: "res4c" name: "res4c_relu" type: "ReLU" } layer { bottom: "res4c" top: "res4d_branch2a" name: "res4d_branch2a" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4d_branch2a" top: "res4d_branch2a" name: "bn4d_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4d_branch2a" top: "res4d_branch2a" name: "scale4d_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4d_branch2a" top: "res4d_branch2a" name: "res4d_branch2a_relu" type: "ReLU" } layer { bottom: "res4d_branch2a" top: "res4d_branch2b" name: "res4d_branch2b" type: "Convolution" convolution_param { num_output: 256 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res4d_branch2b" top: "res4d_branch2b" name: "bn4d_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4d_branch2b" top: "res4d_branch2b" name: "scale4d_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4d_branch2b" top: "res4d_branch2b" name: "res4d_branch2b_relu" type: "ReLU" } layer { bottom: "res4d_branch2b" top: "res4d_branch2c" name: "res4d_branch2c" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4d_branch2c" top: "res4d_branch2c" name: "bn4d_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4d_branch2c" top: "res4d_branch2c" name: "scale4d_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4c" bottom: "res4d_branch2c" top: "res4d" name: "res4d" type: "Eltwise" } layer { bottom: "res4d" top: "res4d" name: "res4d_relu" type: "ReLU" } layer { bottom: "res4d" top: "res4e_branch2a" name: "res4e_branch2a" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4e_branch2a" top: "res4e_branch2a" name: "bn4e_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4e_branch2a" top: "res4e_branch2a" name: "scale4e_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4e_branch2a" top: "res4e_branch2a" name: "res4e_branch2a_relu" type: "ReLU" } layer { bottom: "res4e_branch2a" top: "res4e_branch2b" name: "res4e_branch2b" type: "Convolution" convolution_param { num_output: 256 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res4e_branch2b" top: "res4e_branch2b" name: "bn4e_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4e_branch2b" top: "res4e_branch2b" name: "scale4e_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4e_branch2b" top: "res4e_branch2b" name: "res4e_branch2b_relu" type: "ReLU" } layer { bottom: "res4e_branch2b" top: "res4e_branch2c" name: "res4e_branch2c" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 st

ride: 1 bias_term: false } } layer { bottom: "res4e_branch2c" top: "res4e_branch2c" name: "bn4e_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4e_branch2c" top: "res4e_branch2c" name: "scale4e_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4d" bottom: "res4e_branch2c" top: "res4e" name: "res4e" type: "Eltwise" } layer { bottom: "res4e" top: "res4e" name: "res4e_relu" type: "ReLU" } layer { bottom: "res4e" top: "res4f_branch2a" name: "res4f_branch2a" type: "Convolution" convolution_param { num_output: 256 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4f_branch2a" top: "res4f_branch2a" name: "bn4f_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4f_branch2a" top: "res4f_branch2a" name: "scale4f_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4f_branch2a" top: "res4f_branch2a" name: "res4f_branch2a_relu" type: "ReLU" } layer { bottom: "res4f_branch2a" top: "res4f_branch2b" name: "res4f_branch2b" type: "Convolution" convolution_param { num_output: 256 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res4f_branch2b" top: "res4f_branch2b" name: "bn4f_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4f_branch2b" top: "res4f_branch2b" name: "scale4f_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4f_branch2b" top: "res4f_branch2b" name: "res4f_branch2b_relu" type: "ReLU" } layer { bottom: "res4f_branch2b" top: "res4f_branch2c" name: "res4f_branch2c" type: "Convolution" convolution_param { num_output: 1024 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res4f_branch2c" top: "res4f_branch2c" name: "bn4f_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res4f_branch2c" top: "res4f_branch2c" name: "scale4f_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4e" bottom: "res4f_branch2c" top: "res4f" name: "res4f" type: "Eltwise" } layer { bottom: "res4f" top: "res4f" name: "res4f_relu" type: "ReLU" } layer { bottom: "res4f" top: "res5a_branch1" name: "res5a_branch1" type: "Convolution" convolution_param { num_output: 2048 kernel_size: 1 pad: 0 stride: 2 bias_term: false } } layer { bottom: "res5a_branch1" top: "res5a_branch1" name: "bn5a_branch1" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5a_branch1" top: "res5a_branch1" name: "scale5a_branch1" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res4f" top: "res5a_branch2a" name: "res5a_branch2a" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 2 bias_term: false } } layer { bottom: "res5a_branch2a" top: "res5a_branch2a" name: "bn5a_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5a_branch2a" top: "res5a_branch2a" name: "scale5a_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5a_branch2a" top: "res5a_branch2a" name: "res5a_branch2a_relu" type: "ReLU" } layer { bottom: "res5a_branch2a" top: "res5a_branch2b" name: "res5a_branch2b" type: "Convolution" convolution_param { num_output: 512 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res5a_branch2b" top: "res5a_branch2b" name: "bn5a_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5a_branch2b" top: "res5a_branch2b" name: "scale5a_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5a_branch2b" top: "res5a_branch2b" name: "res5a_branch2b_relu" type: "ReLU" } layer { bottom: "res5a_branch2b" top: "res5a_branch2c" name: "res5a_branch2c" type: "Convolution" convolution_param { num_output: 2048 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res5a_branch2c" top: "res5a_branch2c" name: "bn5a_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5a_branch2c" top: "res5a_branch2c" name: "scale5a_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5a_branch1" bottom: "res5a_branch2c" top: "res5a" name: "res5a" type: "Eltwise" } layer { bottom: "res5a" top: "res5a" name: "res5a_relu" type: "ReLU" } layer { bottom: "res5a" top: "res5b_branch2a" name: "res5b_branch2a" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res5b_branch2a" top: "res5b_branch2a" name: "bn5b_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5b_branch2a" top: "res5b_branch2a" name: "scale5b_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5b_branch2a" top: "res5b_branch2a" name: "res5b_branch2a_relu" type: "ReLU" } layer { bottom: "res5b_branch2a" top: "res5b_branch2b" name: "res5b_branch2b" type: "Convolution" convolution_param { num_output: 512 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res5b_branch2b" top: "res5b_branch2b" name: "bn5b_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5b_branch2b" top: "res5b_branch2b" name: "scale5b_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5b_branch2b" top: "res5b_branch2b" name: "res5b_branch2b_relu" type: "ReLU" } layer { bottom: "res5b_branch2b" top: "res5b_branch2c" name: "res5b_branch2c" type: "Convolution" convolution_param { num_output: 2048 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res5b_branch2c" top: "res5b_branch2c" name: "bn5b_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5b_branch2c" top: "res5b_branch2c" name: "scale5b_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5a" bottom: "res5b_branch2c" top: "res5b" name: "res5b" type: "Eltwise" } layer { bottom: "res5b" top: "res5b" name: "res5b_relu" type: "ReLU" } layer { bottom: "res5b" top: "res5c_branch2a" name: "res5c_branch2a" type: "Convolution" convolution_param { num_output: 512 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res5c_branch2a" top: "res5c_branch2a" name: "bn5c_branch2a" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5c_branch2a" top: "res5c_branch2a" name: "scale5c_branch2a" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5c_branch2a" top: "res5c_branch2a" name: "res5c_branch2a_relu" type: "ReLU" } layer { bottom: "res5c_branch2a" top: "res5c_branch2b" name: "res5c_branch2b" type: "Convolution" convolution_param { num_output: 512 kernel_size: 3 pad: 1 stride: 1 bias_term: false } } layer { bottom: "res5c_branch2b" top: "res5c_branch2b" name: "bn5c_branch2b" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5c_branch2b" top: "res5c_branch2b" name: "scale5c_branch2b" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5c_branch2b" top: "res5c_branch2b" name: "res5c_branch2b_relu" type: "ReLU" } layer { bottom: "res5c_branch2b" top: "res5c_branch2c" name: "res5c_branch2c" type: "Convolution" convolution_param { num_output: 2048 kernel_size: 1 pad: 0 stride: 1 bias_term: false } } layer { bottom: "res5c_branch2c" top: "res5c_branch2c" name: "bn5c_branch2c" type: "BatchNorm" batch_norm_param { use_global_stats: true } } layer { bottom: "res5c_branch2c" top: "res5c_branch2c" name: "scale5c_branch2c" type: "Scale" scale_param { bias_term: true } } layer { bottom: "res5b" bottom: "res5c_branch2c" top: "res5c" name: "res5c" type: "Eltwise" } layer { bottom: "res5c" top: "res5c" name: "res5c_relu" type: "ReLU" } layer { bottom: "res5c" top: "pool5" name: "pool5" type: "Pooling" pooling_param { kernel_size: 7 stride: 1 pool: AVE } } layer { bottom: "pool5" top: "fc1000" name: "fc1000" type: "InnerProduct" inner_product_param { num_output: 1000 } } layer { bottom: "fc1000" top: "prob" name: "prob" type: "Softmax" }

下面贴出一个非常简单的10层的残差网络 ，真实环境下 请用res50 res152

```
import tensorflow as tf from collections import namedtuple from math import sqrt import input_data d
ef conv2d(x, n_filters,           k_h=5, k_w=5,           stride_h=2, stride_w=2,           stddev=0.02,
          activation=lambda x: x,           bias=True,           padding='SAME',           name="Conv2D"):
    with tf.variable_scope(name):         w = tf.get_variable(             'w', [k_h, k_w, x.get_shape()[-1], n_filter
s],             initializer=tf.truncated_normal_initializer(stddev=stddev))         conv = tf.nn.conv2d(
            x, w, strides=[1, stride_h, stride_w, 1], padding=padding)         if bias:             b = tf.get_variable(
                'b', [n_filters],                 initializer=tf.truncated_normal_initializer(stddev=stddev))             con
v = conv + b         return activation(conv) def linear(x, n_units, scope=None, stddev=0.02,           activa
tion=lambda x: x):         shape = x.get_shape().as_list()     with tf.variable_scope(scope or "Linear"):
        matrix = tf.get_variable("Matrix", [shape[1], n_units], tf.float32,                                  tf.random_nor
mal_initializer(stddev=stddev))         return activation(tf.matmul(x, matrix)) # %% def residual_networ
k(x, n_outputs,                 activation=tf.nn.relu):     # %%     LayerBlock = namedtuple(         'LayerB
lock', ['num_repeats', 'num_filters', 'bottleneck_size'])     blocks = [LayerBlock(3, 128, 32),               LayerB
lock(3, 256, 64),               LayerBlock(3, 512, 128),               LayerBlock(3, 1024, 256)]     # %%     inpu
t_shape = x.get_shape().as_list()     if len(input_shape) == 2:         ndim = int(sqrt(input_shape[1]))         i
f ndim * ndim != input_shape[1]:             raise ValueError('input_shape should be square')         x = tf.res
hape(x, [-1, ndim, ndim, 1])     # %%     # First convolution expands to 64 channels and downsamples
    net = conv2d(x, 64, k_h=7, k_w=7,                  name='conv1',                  activation=activation)
    # %%     # Max pool and downsampling     net = tf.nn.max_pool(         net, [1, 3, 3, 1], strides=
[1, 2, 2, 1], padding='SAME')     # %%     # Setup first chain of resnets     net = conv2d(net, blocks[0].nu
m_filters, k_h=1, k_w=1,                  stride_h=1, stride_w=1, padding='VALID', name='conv2')     # %%
    # Loop through all res blocks     for block_i, block in enumerate(blocks):         for repeat_i in range(blo
ck.num_repeats):             name = 'block_%d/repeat_%d' % (block_i, repeat_i)             conv = conv2d(ne
t, block.bottleneck_size, k_h=1, k_w=1,                           padding='VALID', stride_h=1, stride_w=1,
                          activation=activation,                           name=name + '/conv_in')             conv = conv
2d(conv, block.bottleneck_size, k_h=3, k_w=3,                           padding='SAME', stride_h=1, strid
e_w=1,                           activation=activation,                           name=name + '/conv_bottleneck')
            conv = conv2d(conv, block.num_filters, k_h=1, k_w=1,                           padding='VALID', strid
e_h=1, stride_w=1,                           activation=activation,                           name=name + '/conv_out')
            net = conv + net         try:             # upscale to the next block size             next_block = block
s[block_i + 1]             net = conv2d(net, next_block.num_filters, k_h=1, k_w=1,                          paddin
g='SAME', stride_h=1, stride_w=1, bias=False,                          name='block_%d/conv_upscale' % bloc
k_i)         except IndexError:             pass     # %%     net = tf.nn.avg_pool(net,                          ksize=
[1, net.get_shape().as_list()[1],                          net.get_shape().as_list()[2], 1],                          stride
s=[1, 1, 1, 1], padding='VALID')     net = tf.reshape(         net,         [-1, net.get_shape().as_list()[1] *
        net.get_shape().as_list()[2] *         net.get_shape().as_list()[3]])     net = linear(net, n_outputs, activat
ion=tf.nn.softmax)     # %%     return net def rsnn():     """Test the resnet on MNIST."""         mnist = in
put_data.read_data_sets('/tmp/data/', one_hot=True)     x = tf.placeholder(tf.float32, [None, 784])
    y = tf.placeholder(tf.float32, [None, 10])     y_pred = residual_network(x, 10)     # %% Define loss/eva
l/training functions     cross_entropy = -tf.reduce_sum(y * tf.log(y_pred))     optimizer = tf.train.AdamOp
timizer().minimize(cross_entropy)     # %% Monitor accuracy     correct_prediction = tf.equal(tf.argma
x(y_pred, 1), tf.argmax(y, 1))     accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))     # %% W
e now create a new session to actually perform the initialization the     # variables:     sess = tf.Session()
    sess.run(tf.initialize_all_variables())     # %% We'll train in minibatches and report accuracy:     batch_si
ze = 50     n_epochs = 5     for epoch_i in range(n_epochs):         # Training         train_accuracy = 0
        for batch_i in range(mnist.train.num_examples // batch_size):             batch_xs, batch_ys = mnist.tr
ain.next_batch(batch_size)             train_accuracy += sess.run([optimizer, accuracy], feed_dict={
                x: batch_xs, y: batch_ys})[1]         train_accuracy /= (mnist.train.num_examples // batch_size)
        # Validation         valid_accuracy = 0         for batch_i in range(mnist.validation.num_examples // b
atch_size):             batch_xs, batch_ys = mnist.validation.next_batch(batch_size)             valid_accurac
y += sess.run(accuracy,                                       feed_dict={                                           x: batch_xs,
                                      y: batch_ys                                       })         valid_accuracy /= (mnist.validat
ion.num_examples // batch_size)         print('epoch:', epoch_i, ', train:',               train_accuracy, ', valid:', v
alid_accuracy) if __name__ == '__main__':     rsnn()
```

```
root@iZu1cdurunpZ:~/tensorflowtest# vi rsnn.py
root@iZu1cdurunpZ:~/tensorflowtest# python rsnn.py
('Extracting', '/tmp/data/train-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
```

下面 是迭代一次的准确率

```
root@iZu1cdurunpZ:~/tensorflowtest# python rsnn.py
('Extracting', '/tmp/data/train-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
('epoch:', 0, ', train:', 0.86889090818099002, ', valid:', 0.96679999887943269)
```

这里我就不等他迭代完成了。。。。

上一篇：TensorFlow人工智能引擎入门教程之九 RNN/LSTM循环神经网络长短期记忆网络使用

下一篇：TensorFlow人工智能入门教程之十一 最强网络DLSTM 双向长短期记忆网络（阿里小AI实现）