从0到1实现基于Tornado和Tensorflow的人脸、年龄、性别识别(2)

时间 2017-03-17

栏目 Python (http://www.voidcn.com/column/python)

原文 http://blog.csdn.net/nanjingdreamfly/article/details/62886909

每日一句

每一个你不满意的现在，都有一个你没有努力的曾经。

年龄识别模型的训练过程

```
def main(argv=None):
    ### 一个图中包含有一个名称范围的堆栈，在使用name_scope(...)之后，将压(push)新名称进栈中，
#并在下文中使用该名称
    with tf.Graph().as_default():

        ##这句比较重要选择模型
        model_fn = select_model(FLAGS.model_type)
        ##假设这里选的是默认的，这里我们发现其实作者是实现了inception v3、levi_hassner_bn两种模型
        ## Open the metadata file and figure out nlabels, and size of epoch
        ## 年龄模型的年龄段labes分别是['(0, 2)','(4, 6)','(8, 12)','(15, 20)','(25, 32)','(38, 4
3)','(48, 53)','(60, 100)'] 应该是8个标签
```

可以看到，select_model选择了模型，我们以def levi_hassner(nlabels, images, pkeep, is_training):为例子，
这个模型的命名是因为Gil Levi and Tal Hassner, Age and Gender Classification using Convolutional Neural Networks, IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG), at the IEEE Conf. on Computer Vision and Pattern

最新文章

Recognition (CVPR), Boston, June 2015这篇文章。
模型的实现源代码


```
weight_decay = 0.0005
    weights_regularizer = tf.contrib.layers.l2_regularizer(weight_decay)
    with tf.variable_scope("LeviHassner", "LeviHassner", [images]) as scope:
        with tf.contrib.slim.arg_scope(
                [convolution2d, fully_connected],
                weights_regularizer=weights_regularizer,
                biases_initializer=tf.constant_initializer(1.),
                weights_initializer=tf.random_normal_initializer(stddev=0.005),
                trainable=True):
            with tf.contrib.slim.arg_scope(
                    [convolution2d],
                    weights_initializer=tf.random_normal_initializer(stddev=0.01)):

                conv1 = convolution2d(images, 96, [7,7], [4, 4], padding='VALID', biases_ini
tializer=tf.constant_initializer(0.), scope='conv1')
                pool1 = max_pool2d(conv1, 3, 2, padding='VALID', scope='pool1')
                norm1 = tf.nn.local_response_normalization(pool1, 5, alpha=0.0001,
beta=0.75, name='norm1')
                conv2 = convolution2d(norm1, 256, [5, 5], [1, 1], padding='SAME', scope='con
v2')
                pool2 = max_pool2d(conv2, 3, 2, padding='VALID', scope='pool2')
                norm2 = tf.nn.local_response_normalization(pool2, 5, alpha=0.0001,
beta=0.75, name='norm2')
                conv3 = convolution2d(norm2, 384, [3, 3], [1, 1], biases_initializer=tf.cons
tant_initializer(0.), padding='SAME', scope='conv3')
                pool3 = max_pool2d(conv3, 3, 2, padding='VALID', scope='pool3')
                flat = tf.reshape(pool3, [-1, 384*6*6], name='reshape')
                full1 = fully_connected(flat, 512, scope='full1')
                drop1 = tf.nn.dropout(full1, pkeep, name='drop1')
                full2 = fully_connected(drop1, 512, scope='full2')
                drop2 = tf.nn.dropout(full2, pkeep, name='drop2')
```

这是典型的卷积神经网络用于分类的模型结构

- 1. 刚刚，吴恩达讲了干货满满的一节全新AI课，全程手写板书 (http://www.voidcn.com/article/p-nogcydpz-boa.html)
- 2. 算法-动态规划 (http://www.voidcn.com/article/p-wwnvrfnk-boa.html)
- 3. 机房收费结束感想 (http://www.voidcn.com/article/p-okqrmydl-boa.html)
- 4. 使用pycaffe读取caffemodel参数 (http://www.voidcn.com/article/p-bujiiols-boa.html)
- 5. Ubuntu下vim+ctags的配置 (http://www.voidcn.com/article/p-slxmiatd-boa.html)
- 6. 【Angular】——localStorage和sessionStorage (http://www.voidcn.com/article/p-taravojb-boa.html)
- 7. win10 uwp 列表模板选择器 (http://www.voidcn.com/article/p-mvgratcs-boa.html)
- 8. win10 uwp 后台获取资源 (http://www.voidcn.com/article/p-byjnrjch-boa.html)
- 9. jsp字符编码过滤器的实现 (http://www.voidcn.com/article/p-ydvnmlyo-boa.html)
- 10. Nginx安装 (http://www.voidcn.com/article/p-hybiuzxm-boa.html)

相关文章

- 1. TensorFlow通过人脸识别年龄（2）

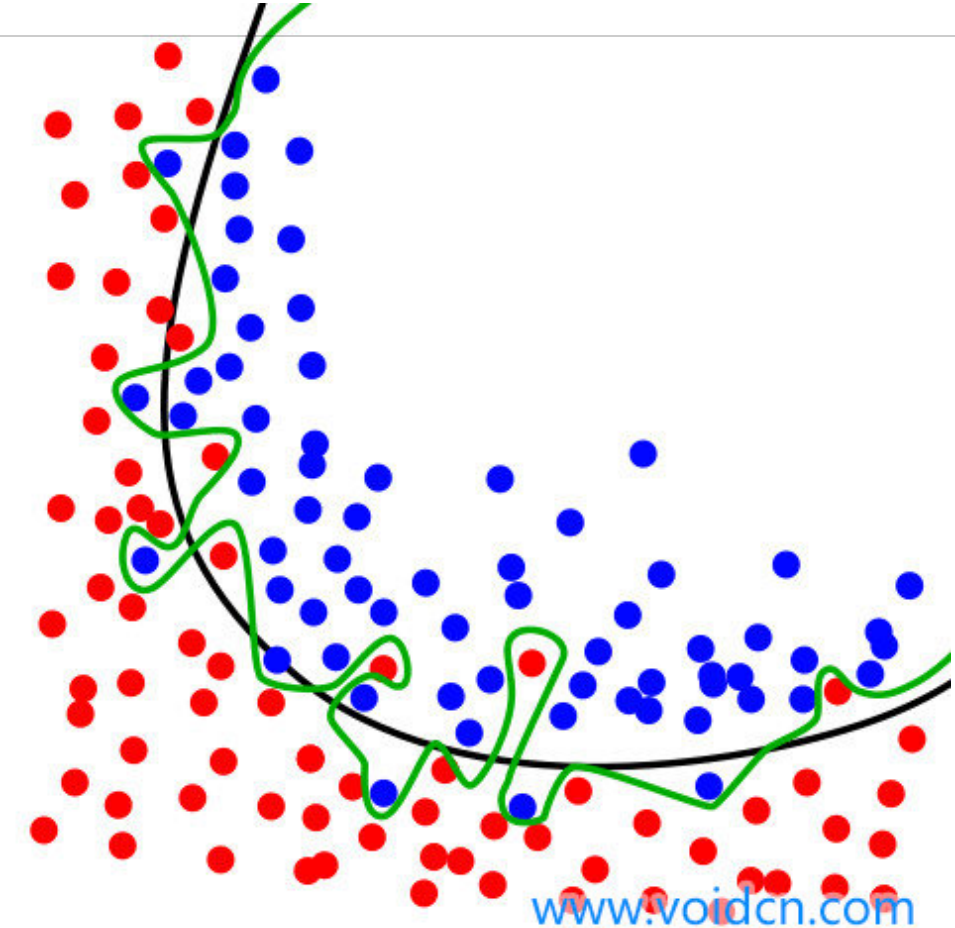
L2范数

其中 weights_regularizer = tf.contrib.layers.l2_regularizer(weight_decay)

l2_regularizer是L2范数的意思，在《线性代数》《矩阵论中》，我们还了解到有L1范数，那么更受宠幸的规则化范数是L2范数: ||W||2。它也不逊于L1范数，它有两个美称，在回归里面，有人把有它的回归叫“岭回归”（Ridge Regression），有人也叫它“权值衰减weight decay”。

它的强大功效是改善机器学习里面一个非常重要的问题：过拟合。至于过拟合是什么，上面也解释了，就是模型训练时候的误差很小，但在测试的时候误差很大，也就是我们的模型复杂到可以拟合到我们的所有训练样本了，但在实际预测新的样本的时候，糟糕的一塌糊涂。通俗的讲就是应试能力很强，实际应用能力很差。

下面这幅图可以看到过拟合大概是什么状况，绿色线



在上面的图像中有两个不同的类，分别由蓝色和红色圆圈表示。绿线是过度拟合的分类器。它完全遵循训练数据，同时也严重依赖于训练数据，并且可能在处理未知数据时比代表正则化模型的黑线表现更差。因此，我们的正则化目标是得到一个简单的模

(http://www.voidcn.com/article/p-pfjqbogs-oo.html)

- 2. TensorFlow通过人脸识别年龄（1） (http://www.voidcn.com/article/p-bgmvvasj-oo.html)
- 3. 基于CNN的人脸 性别、年龄识别 (http://www.voidcn.com/article/p-hnlntogv-ty.html)
- 4. 基于caffe的性别、年龄识别 (http://www.voidcn.com/article/p-gknctojw-pb.html)
- 5. 基于CNN的性别、年龄识别 (http://www.voidcn.com/article/p-yhfhnaht-bmd.html)
- 6. 人脸和性别识别（基于OpenCV） (http://www.voidcn.com/article/p-hqkjcbch-nm.html)
- 7. android人脸识别——HowOld测测你的年龄和性别 (http://www.voidcn.com/article/p-ebvjxoyi-uk.html)
- 8. 基于opencv的人脸性别识别 (http://www.voidcn.com/article/p-qkfwsweb-bde.html)
- 9. 基于CNN的性别、年龄识别及Demo实现 (http://www.voidcn.com/article/p-cmlyhwbo-xp.html)
- 10. 人脸属性识别算法 | 性别+种族+年龄+表情 (http://www.voidcn.com/article/p-uvnqvipt-oo.html)

>>更多相关文章<<
(http://www.voidcn.com/relative/p-qhajwjye-oo.html)

型，不附带任何不必要的复杂。我们选择L2-正则化来实现这一点，L2正则化将网络中所有权重的平方和加到损失函数。如果模型使用大权重，则对应重罚分，并且如果模型使用小权重，则小罚分。
这就是为什么我们在定义权重时使用了regularizer参数，并为它分配了一个l2_regularizer。这告诉了TensorFlow要跟踪l2_regularizer这个变量的L2正则化项（并通过参数reg_constant对它们进行加权）。所有正则化项被添加到一个损失函数可以访问的集合——tf.GraphKeys.REGULARIZATION_LOSSES。

> *将所有正则化损失的总和与先前计算的交叉熵相加，以得到我们的模型的总损失。*

理解是：限制了参数很小，实际上就限制了多项式某些分量的影响很小，这样就相当于减少参数个数，过拟合状况就会被降低。
下面走到 with tf.variable_scope(“LeviHassner”, “LeviHassner”, [images]) as scope:
可以看到有不少scope，那么它是什么呢？
Tensorflow 为了更好的管理变量,提供了variable scope机制，具体参考TensorFlow官方这篇文章
https://www.tensorflow.org/programmers_guide/variable_scope
后面又来了个比较复杂的东西 tf.contrib.slim，这是什么鬼，看看官方的解释：
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim
TF-Slim is a lightweight library for defining, training and evaluating complex models in TensorFlow. Components of tf-slim can be freely mixed with native tensorflow, as well as other frameworks, such as tf.contrib.learn.
我们用到的是arg_scope
arg_scope: provides a new scope named arg_scope that allows a user to define default arguments for specific operations within that scope.

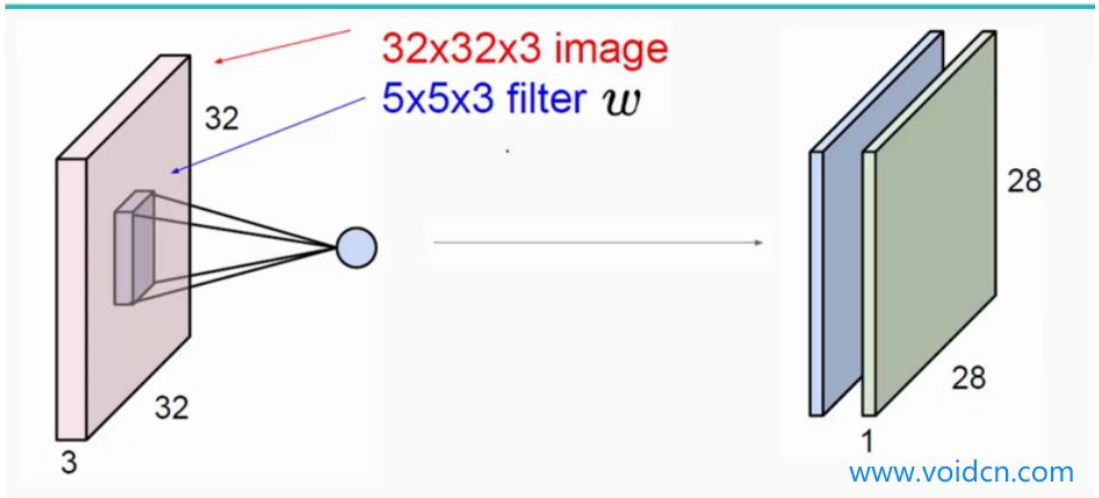
卷积神经网络

卷积神经网络（CNN）由输入层、卷积层、激活函数、池化层、全连接层组成，即INPUT-CONV-RELU-POOL-FC
下面终于来到网络层了啊！来看代码

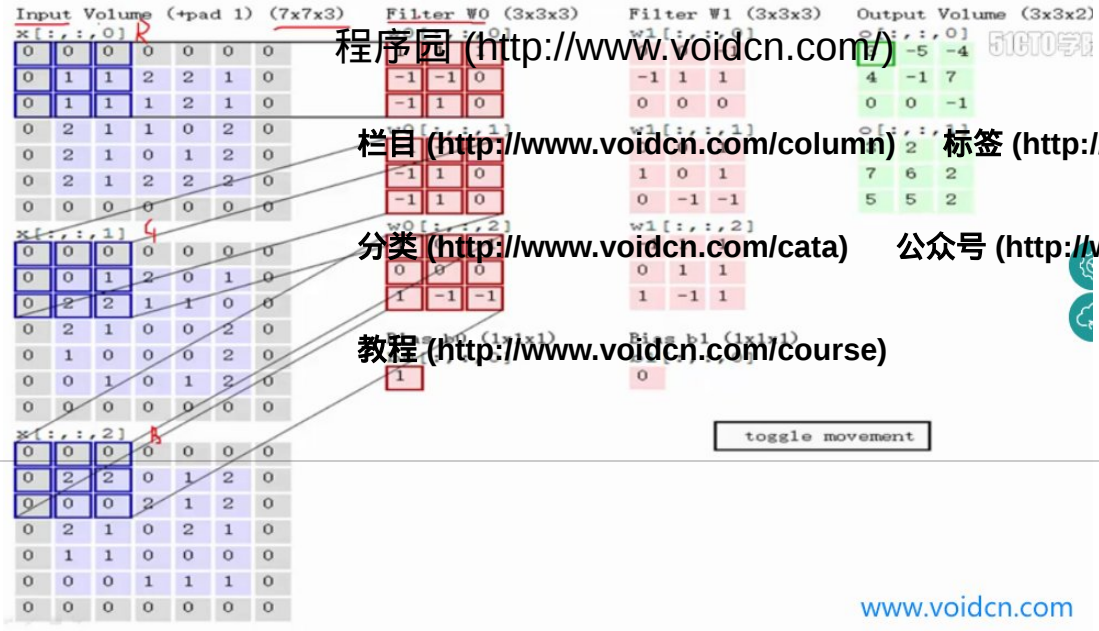
```
conv1 = convolution2d(images, 96, [7,7], [4, 4], padding='VALID', biases_initializer=tf.cons
tant_initializer(0.), scope='conv1')
                pool1 = max_pool2d(conv1, 3, 2, padding='VALID', scope='pool1')
                conv2 = convolution2d(pool1, 256, [5, 5], [1, 1], padding='SAME', scope='con
v2')
                pool2 = max_pool2d(conv2, 3, 2, padding='VALID', scope='pool2')
                conv3 = convolution2d(pool2, 384, [3, 3], [1, 1], padding='SAME', biases_ini
tializer=tf.constant_initializer(0.), scope='conv3')
                pool3 = max_pool2d(conv3, 3, 2, padding='VALID', scope='pool3')
                # can use tf.contrib.layer.flatten
                flat = tf.reshape(pool3, [-1, 384*6*6], name='reshape')
                full1 = fully_connected(flat, 512, scope='full1')
                drop1 = tf.nn.dropout(full1, pkeep, name='drop1')
                full2 = fully_connected(drop1, 512, scope='full2')
                drop2 = tf.nn.dropout(full2, pkeep, name='drop2')
```

卷积层

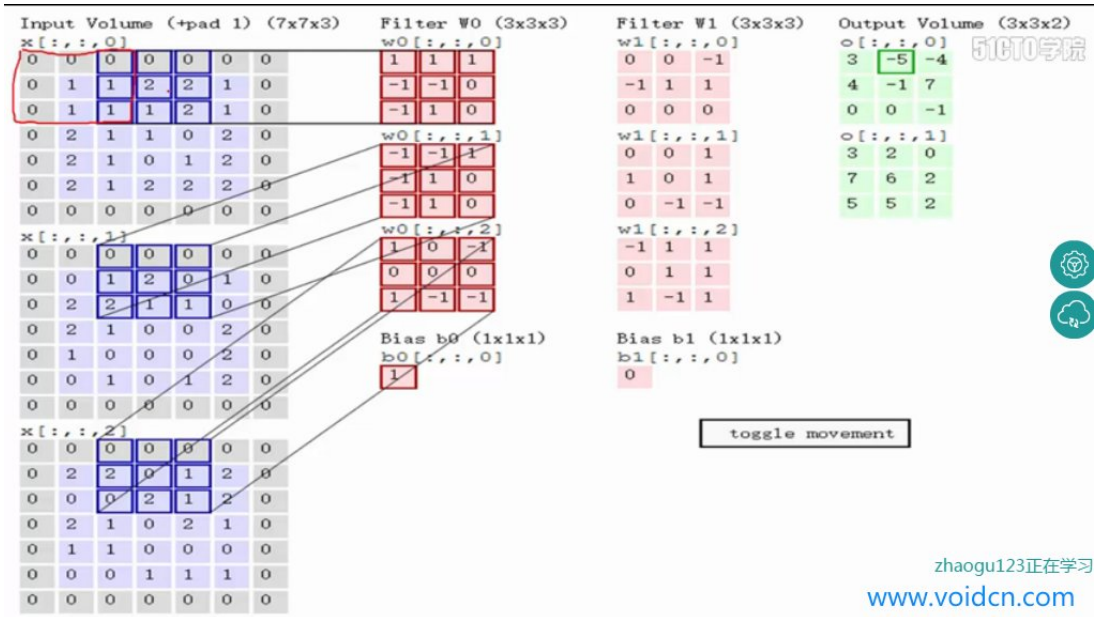
卷积层：用它来进行特征提取，如下：



关于卷积的图解如下：



输入图像和filter的对应位置元素相乘再求和，最后再加上b,得到特征图。如图中所示，filter w0的第一层深度和输入图像的蓝色方框中对应元素相乘再求和得到0，其他两个深度得到2，0，则有0+2+0+1=3即图中右边特征图的第一个元素3.，卷积过后输入图像的蓝色方框再滑动，stride=2，如下：



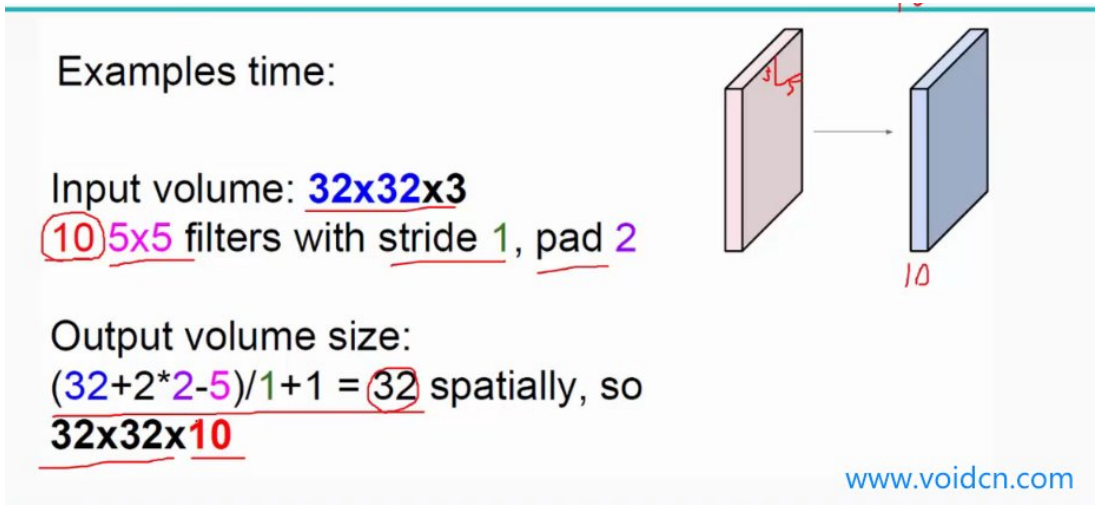
如上图，完成卷积，得到一个3*3*1的特征图，在这里还要注意一点，即zero pad项，即为图像加上一个边界，边界元素均为0.（对原输入无影响）一般有

F=3 => zero pad with 1

F=5 => zero pad with 2

F=7=> zero pad with 3,边界宽度是一个经验值，加上zero pad这一项是为了使输入图像和卷积后的特征图具有相同的维度，如：输入为5*5*3，filter为3*3*3，在zero pad 为1，则加上zero pad后的输入图像为7*7*3，则卷积后的特征图大小为5*5*1（（7-3）/1+1），与输入图像一样；

卷积层还有一个特性就是“权值共享”原则。如下图：



如没有这个原则，则特征图由10个32*32*1的特征图组成，即每个特征图上有1024个神经元，每个神经元对应输入图像上一块5*5*3的区域，即一个神经元和输入图像的这块区域有75个连接，即75个权值参数，则共有75*1024*10=768000个权值参数，这是非常复杂的，因此卷积神经网络引入“权值”共享原则，即一个特征图上每个神经元对应的75个权值参数被每个神经元共享，这样则只需75*10=750个权值参数，而每个特征图的阈值也共享，即需要10个阈值，则总共需要750+10=760个参数。

而关于特征图的大小计算方法具体如下：

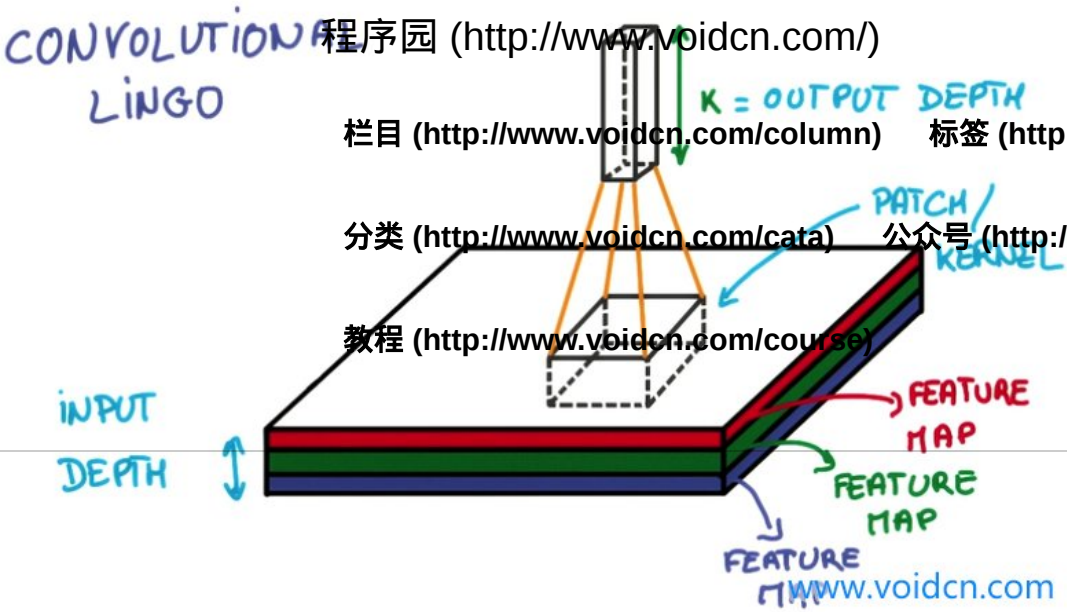
第一层是个卷积层convolution2d(images, 96, [7,7], [4, 4], padding='VALID', biases_initializer=tf.constant_initializer(0.), scope='conv1')，

tf.contrib.layers.convolution2d(*args, **kwargs)

通过pycharm跟到tensorflow源码，其实是调用了下面这个函数


```
def convolution(inputs,
num_outputs,
kernel_size,
stride=1,
padding='SAME',
data_format=None,
rate=1,
activation_fn=nn.relu,
normalizer_fn=None,
normalizer_params=None,
weights_initializer=initializers.xavier_initializer(),
weights_regularizer=None,
biases_initializer=init_ops.zeros_initializer(),
biases_regularizer=None,
reuse=None,
variables_collections=None,
outputs_collections=None,
trainable=True,
scope=None):
Returns:
a tensor representing the output of the operation.
```

有点难以理解是吧，还是稍微再回顾一下卷积吧！



这个函数很强大，1到3维的卷积都支持。(这里暂时只用2维)

· inputs: 输入变量，是一个N+2维的Tensor

类型要求是一个Tensor，而我们一般训练的数据都是常量(比如mnist，load以后得到是python的数据类型，不是tf的)，所以需要把用tf的方法做一下转换，比如tf.reshape

为什么是N+2维呢，比如图像，除了宽度和高度，实际上还有样本数量和通道数量(如RGB3通道)，所以多了2维。

inputs的格式，由date_format这个参数来觉得，比如2维，有NHWC和NCHW两种。N是样本数量，H高度，W宽度，C通道数。

· num_outputs: 卷积filter的数量，或者说提取的特征数量，比如5,10

· kernel_size: 卷积核的大小，是N个参数的list，比如二维图像，可以时候[10,10]，如果参数值相同，用一个整数来· 表示也可以；

· stride: 卷积步长，同样是N个参数的序列，或者都相等的话，用一个整数来表示，默认是1.

· padding: 字符串格式，默认SAME，可选'VALID'。（我也不知道效果差异怎么样，具体的滤波过程，看下面的图）

· data_format: 字符串，指定inputs的格式

一维数据："NWC" (default) and "NCW"

二维数据："NHWC" (default) and "NCHW"

三维数据："NDHWC"

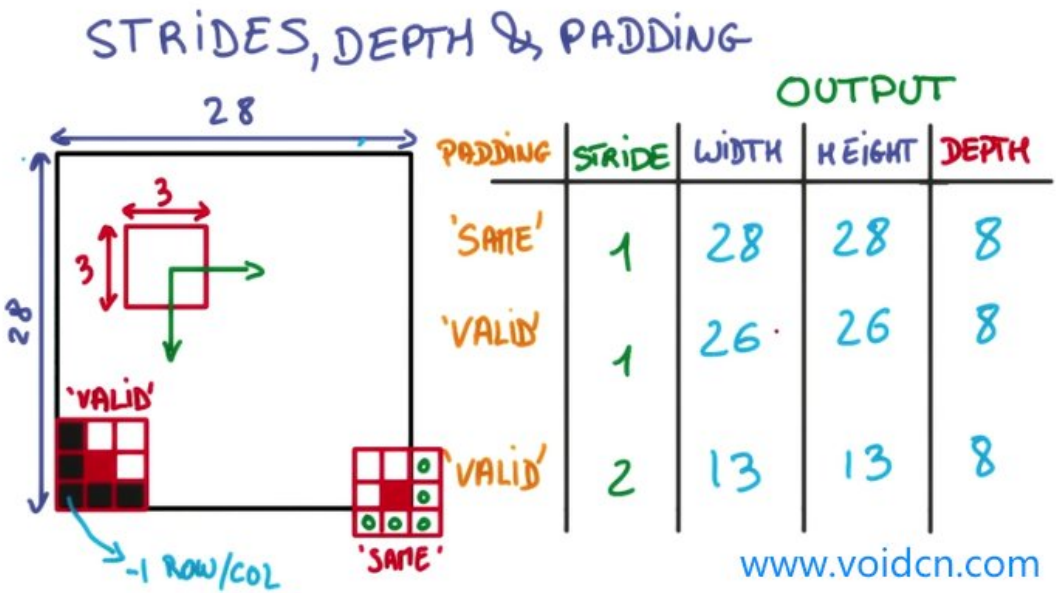
也就是，不指定的话，通道数都是最后一个参数。

· rate: a sequence of N positive integers specifying the dilation rate to use for a'trous convolution. Can be a single integer to specify the same value for all spatial dimensions. (暂时没看到其作用)

· activation_fn: 激活函数，默认relu

· normalizer_fn: normalization function to use instead of biases.（没用过，不知道起作用）

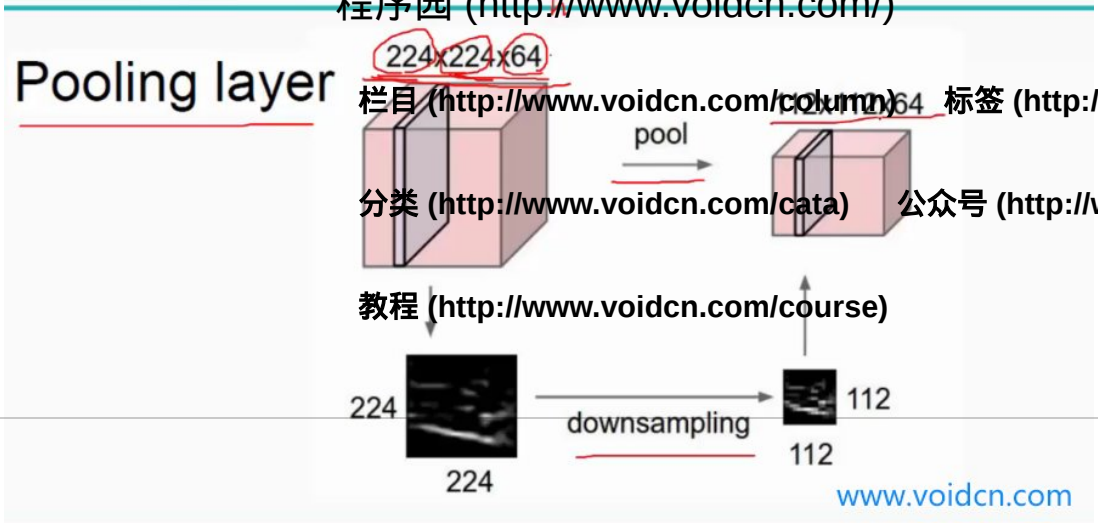
· normalizer_params： normalization function parameters.
· weights_initializer： 这不用说了，有默认值，估计用默认的就可以了。
· weights_regularizer: Optional regularizer for the weights.(防止过拟合)
· biases_initializer: 有默认值，一般也都不用指定。
· biases_regularizer: …
· reuse: whether or not the layer and its variables should be reused. To be able to reuse the layer scope · must be given. 应该都需要reuse吧，所以这个参数默认为True更好，现在是None。
· variables_collections： 怎么用暂时不太明白，但应该不用指定也可以；
· outputs_collections： 同上，
· trainable： If True also add variables to the graph collection GraphKeys.TRAINABLE_VARIABLES，默认是True。（这个是不是说在fit的时候需要设为True，evaluate和predict的时候为false？）
· scope： 也即是variable_scope， 如果用多个卷积层的话，需要设置这个参数，以便把每一次的weight和bias区别出来。



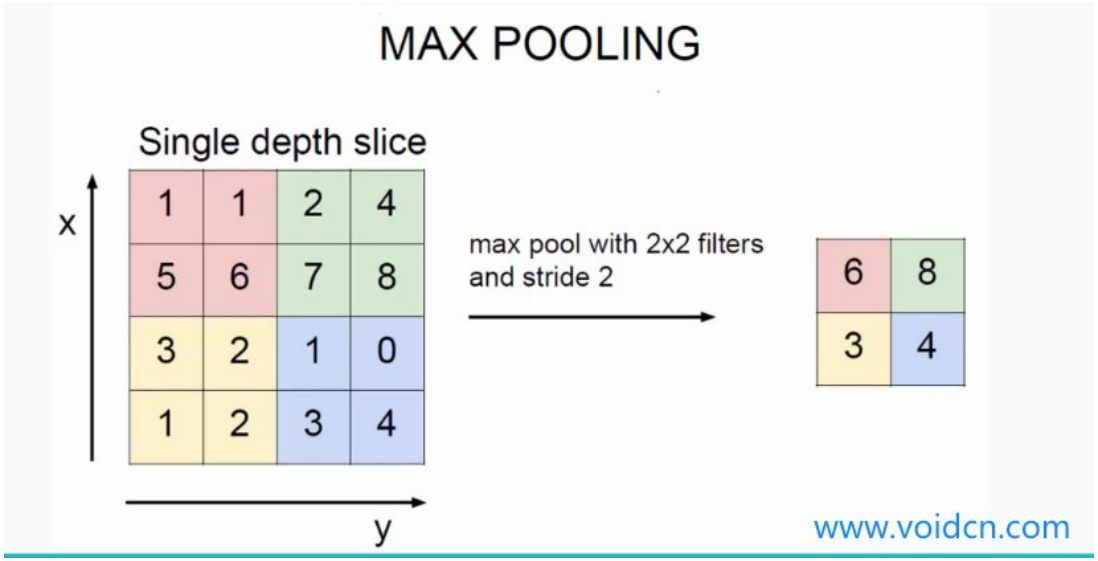
了解更多关于layer的函数说明，访问tensorflow官方网站
https://www.tensorflow.org/api_guides/python/contrib.layers

池化层

池化层：对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征



池化操作一般有两种，一种是Avy Pooling,一种是max Pooling,如下：



同样地采用一个2*2的filter,max pooling是在每一个区域中寻找最大值，这里的stride=2,最终在原特征图中提取主要特征得到右图。

（Avy pooling现在不怎么用了，方法是对每一个2*2的区域元素求和，再除以4，得到主要特征），而一般的filter取2*2,最大取3*3,stride取2，压缩为原来的1/4.

注意：这里的pooling操作是特征图缩小，有可能影响网络的准确度，因此可以通过增加特征图的深度来弥补（这里的深度变为原来的2倍）。

可以用 tf.contrib.layers.max_pool2d或者tf.contrib.layers.avg_pool2d


```
max_pool2d(inputs, kernel_size, stride=2, padding='VALID', data_format=DATA_FORMAT_NHWC,
outputs_collections=None, scope=None)
```


inputs: 就是卷积的输出了；
kernel_size: 是不是叫pool_size更贴切。[kernel_height, kernel_width]或者是一个整数，
stride： [stride_height, stride_width]，不过文档上说目前这两个值必须一样
padding： 这里默认是VALID，和卷积默认不一样
data_format: 注意和卷积用的一样哦；
outputs_collections: …
scope: pooling的时候没有参数，需要scope吗？

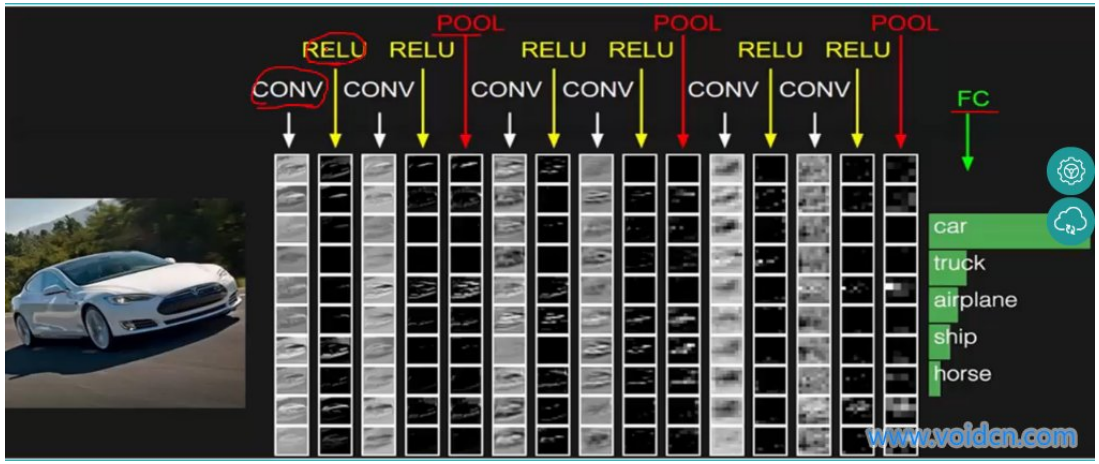
reshape

```
flat = tf.reshape(pool3, [-1, 384*6*6], name='reshape')
```

-1，就是缺省值，就是先以你们合适，到时总数除以你们几个的乘积，我该是几就是几。
Reshape output to fit fully connected layer input

全连接层

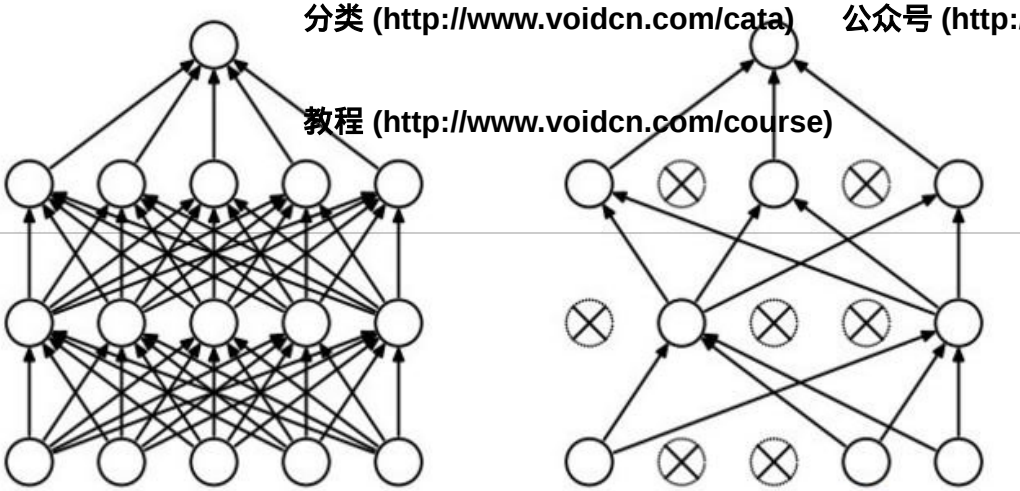
连接所有的特征，将输出值送给分类器（如softmax分类器）。其实是神经网络中的隐含层。



dropout

```
drop2 = tf.nn.dropout(full2, pkeep, name='drop2')
```


这里的keep_prob是保留概率，即我们要保留的RELU的结果所占比例，tensorflow建议的语法是，让它作为一个placeholder，在run时传入。
大概的意思如下图所示：



dropout一般用在全连接的部分，卷积部分不会用到dropout,输出曾也不会使用dropout，适用范围[输入，输出)
http://www.voidcn.com/article/p-hoxjchoi-od.html (http://www.voidcn.com/article/p-hoxjchoi-od.html)

上面只是说了深度学习的网络层，下面讲一下训练方法：

训练方法

```
logits = model_fn(md['nlabels'], images, 1-FLAGS.pdrop, True)
        total_loss = loss(logits, labels)
        train_op = optimizer(FLAGS.optim, FLAGS.eta, total_loss)
        saver = tf.train.Saver(tf.global_variables())
        summary_op = tf.summary.merge_all()
```

model_fn就是之前的网络层模型，loss是误差函数，optimzer是优化方法，先看下误差函数

误差函数


```
def loss(logits, labels):
    labels = tf.cast(labels, tf.int32)
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
        logits=logits, labels=labels, name='cross_entropy_per_example')
    cross_entropy_mean = tf.reduce_mean(cross_entropy, name='cross_entropy')
    tf.add_to_collection('losses', cross_entropy_mean)
    losses = tf.get_collection('losses')
    regularization_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
    total_loss = cross_entropy_mean + LAMBDA * sum(regularization_losses)
    tf.summary.scalar('tl (raw)', total_loss)
    #total_loss = tf.add_n(losses + regularization_losses, name='total_loss')
    loss_averages = tf.train.ExponentialMovingAverage(0.9, name='avg')
    loss_averages_op = loss_averages.apply(losses + [total_loss])
    for l in losses + [total_loss]:
        tf.summary.scalar(l.op.name + ' (raw)', l)
        tf.summary.scalar(l.op.name, loss_averages.average(l))
    with tf.control_dependencies([loss_averages_op]):
        total_loss = t
```

为了使用Tensorboard来可视化我们的数据，我们会经常使用Summary，最终都会用一个简单的merge_all函数来管理我们的Summary

sparse_softmax_cross_entropy_with_logits这个函数的作用就是计算最后一层是softmax层的cross entropy，只不过tensorflow把softmax计算与cross entropy计算放到一起了，用一个函数来实现，用来提高程序的运行速度，原话就是

reduce_mean

tensorflow中有一类在tensor的某一维度上求值的函数。如：

求最大值tf.reduce_max(input_tensor, reduction_indices=None, keep_dims=False, name=None)

求平均值tf.reduce_mean(input_tensor, reduction_indices=None, keep_dims=False, name=None)

tf.reduce_mean(x) 在所有的元素中取平均值

可以看一下total_loss的计算方法，印证了在L2范数那一节中说的：

> *将所有正则化损失的总和与先前计算的交叉熵相加，以得到我们的模型的总损失。*

优化方法


```
def optimizer(optim, eta, loss_fn):
    global_step = tf.Variable(0, trainable=False)
    optz = optim
    if optim == 'Adadelta':
        optz = lambda lr: tf.train.AdadeltaOptimizer(lr, 0.95, 1e-6)
        lr_decay_fn = None
    elif optim == 'Momentum':
        optz = lambda lr: tf.train.MomentumOptimizer(lr, MOM)
        lr_decay_fn = exponential_staircase_decay()

    return tf.contrib.layers.optimize_loss(loss_fn, global_step, eta, optz,
clip_gradients=4., learning_rate_decay_fn=lr_decay_fn)
```

可以看到用了python的lambda匿名函数，lambda 并不会带来程序运行效率的提高，只会使代码更简洁。
本程序中用了两种优化器
AdadeltaOptimizer
MomentumOptimizer

操作	描述
tf.train.Optimizer.get_slot_names()	返回一个由Optimizer所创建的slots的名称列表
tf.train.Optimizer.get_slot(var, name)	返回一个name所对应的slot，name是由Optimizer为var所创建
class tf.train.GradientDescentOptimizer	使用梯度下降算法的Optimizer
class tf.train.AdadeltaOptimizer	使用Adadelta算法的Optimizer
tf.train.AdagradOptimizer	创建Adagrad优化器
class tf.train.MomentumOptimizer	使用Momentum算法的Optimizer

优化器的科普介绍可以参考以下视频教程
《莫烦 Tensorflow 13 优化器 optimizer (神经网络 教学教程tutorial)_标清》，或者看以下链接
http://int8.io/comparison-of-optimization-techniques-stochastic-gradient-descent-momentum-adagrad-and-adadelta/
In Image below, we see their behavior on the contours of a loss surface over time. Note that Adagrad, Adadelta, and RMSprop almost immediately head off in the right direction and converge similarly fast, while Momentum and NAG are led off-track, evoking the image of a ball rolling down the hill. NAG, however, is quickly able to correct its course due to its increased responsiveness by looking ahead and heads to the minimum.

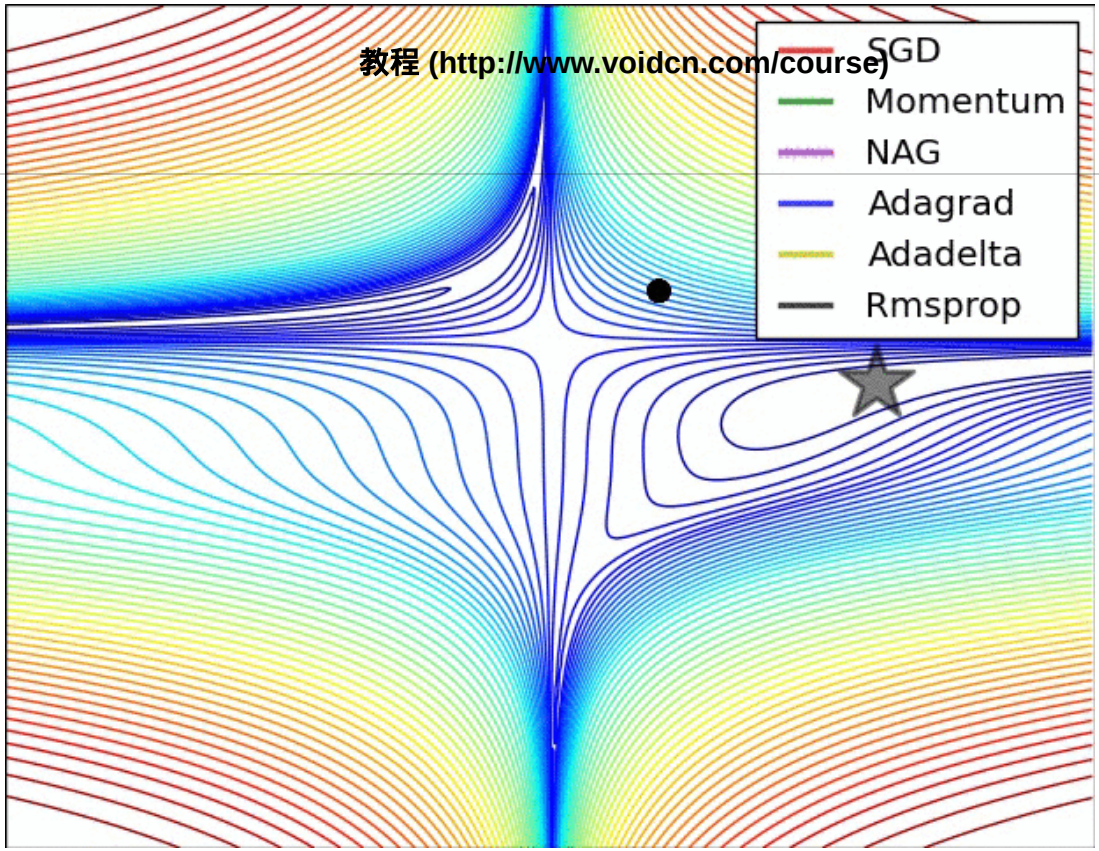
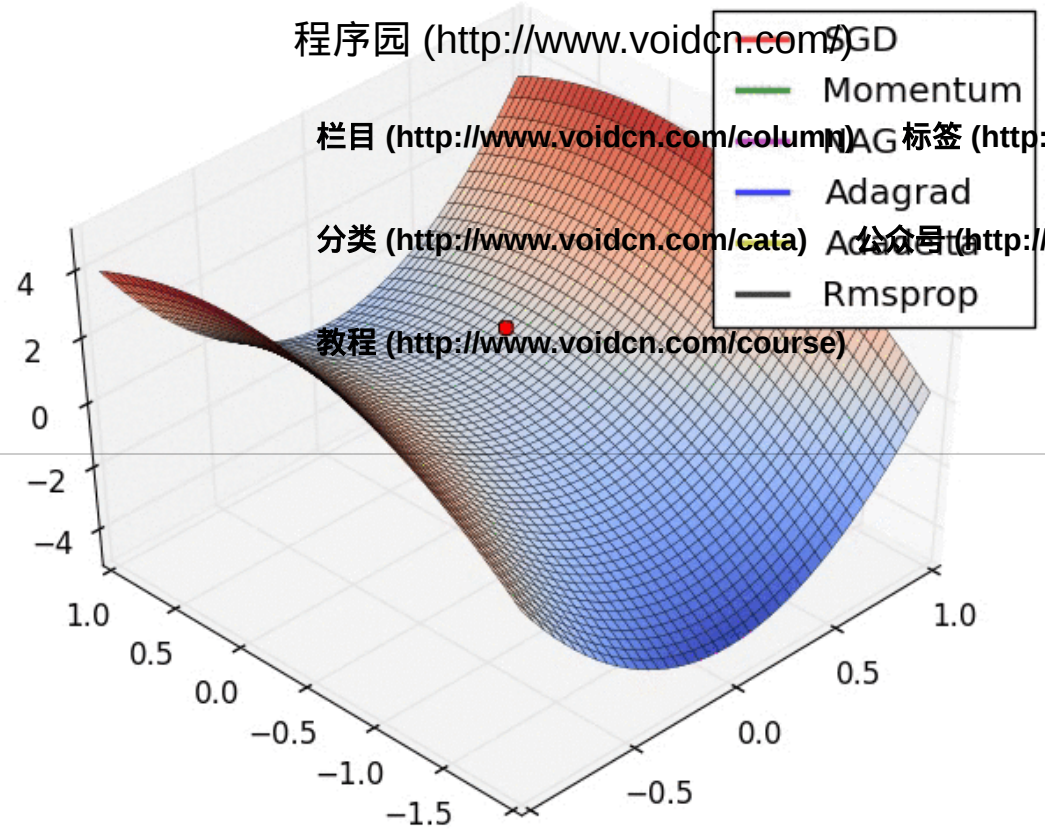


Image blow shows the behaviour of the algorithms at a saddle point, i.e. a point where one dimension has a positive slope, while the other dimension has a negative slope, which pose a difficulty for SGD as we mentioned before. Notice here that SGD, Momentum, and NAG find it difficulty to break symmetry, although the two latter eventually manage to escape the saddle point, while Adagrad, RMSprop, and Adadelta quickly head down the negative slope.



alphaGo用了RmsProp优化算法
后面是实际的训练过程和打印，没有什么可说的

```
for step in xrange(num_steps):
            start_time = time.time()
            _, loss_value = sess.run([train_op, total_loss])
            duration = time.time() - start_time

            assert not np.isnan(loss_value), 'Model diverged with loss = NaN'

            if step % 10 == 0:
                num_examples_per_step = FLAGS.batch_size
                examples_per_sec = num_examples_per_step / duration
                sec_per_batch = float(duration)

                format_str = ('%s: step %d, loss = %.3f (%.1f examples/sec; %.3f ' 'sec/batc
h)')
                print(format_str % (datetime.now(), step, loss_value,
                                    examples_per_sec, sec_per_batch))

            # Loss only actually evaluated every 100 steps?
            if step % 100 == 0:
                summary_str = sess.run(summary_op)
                summary_writer.add_summary(summary_str, step)

            if step % 1000 == 0 or (step + 1) == num_steps:
                saver.save(sess, checkpoint_path, global_step=step)
```

年龄段识别的预测过程

```
def guessAge(image_file):

    #import!!!Fix the bug http://stackoverflow.com/questions/33765336/remove-nodes-from-grap
h-or-reset-entire-default-graph
    tf.reset_default_graph()
    with tf.Session() as sess:

        age_label_list = AGE_LIST
        agelabels = len(age_label_list)

        # print('Executing on %s' % FLAGS.device_id)
        model_fn = select_model('')

        images = tf.placeholder(tf.float32, [None, RESIZE_FINAL, RESIZE_FINAL, 3])
        logits_age = model_fn(agelabels, images, 1, False)
        init = tf.global_variables_initializer()

        requested_step = FLAGS.requested_step if FLAGS.requested_step else None

        checkpoint_path = '%s' % (AGE_MODEL_PATH)

        model_checkpoint_path, global_step = get_checkpoint(checkpoint_path, requested_step,
 FLAGS.checkpoint)

        saver = tf.train.Saver()
        if not saver.last_checkpoints :
            saver.restore(sess, model_checkpoint_path)


        softmax_output = tf.nn.softmax(logits_age)

        coder = ImageCoder()

        files = []

        # detect age
```

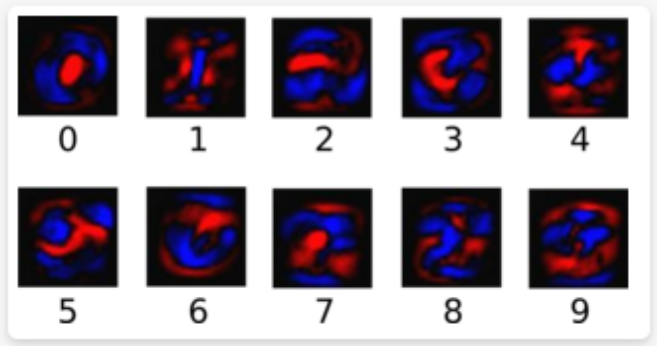
```
        best_choice = classify(sess, age_label_list, softmax_output, coder, images, image_fi
le)


        sess.close()
        return best_choice
```


可以看到 logits_age = model_fn(agelabels, images, 1, False)实现了分类，后面通过 softmax_output = tf.nn.softmax(logits_age)进行softmax归一化，
softmax经常出现，让我们了解一下他的意思吧！

Softmax回归

softmax回归（softmax regression）分两步：第一步
为了得到一张给定图片属于某个特定数字类的证据（evidence），我们对图片像素值进行加权求和。如果这个像素具有很强的证据说明这张图片不属于该类，那么相应的权值为负数，相反如果这个像素拥有有利的证据支持这张图片属于这个类，那么权值是正数。
下面的图片显示了一个模型学习到的图片上每个像素对于特定数字类的权值。红色代表负数权值，蓝色代表正数权值。



我们也需要加入一个额外的偏置量（bias），因为输入往往会带有一些无关的干扰量。因此对于给定的输入图片x它代表的是数字i的证据可以表示为

$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

其中Wi代表权重，bi 代表数字 i 类的偏置量，j 代表给定图片 x 的像素索引用于像素求和。然后用softmax函数可以把这些证据转换成概率 y：

$$y = \text{softmax}(\text{evidence})$$

这的softmax可是看做是一个sigmoid形式的函数，把我们定义的线性函数的输出转换成我们想要的格式，也就是关于10个数字类的概率分布。因此，给定一张图片，它对于每一个数字的吻合度可以被softmax函数转换成为一个概率值。

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

展开等式右边的子式，可以得到：

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

可以看到softmax回归处理后，经过classify来判断年龄，跟进classify函数

```
def classify(sess, label_list, softmax_output, coder, images, image_file):

    print('Running file %s' % image_file)
    image_batch = make_batch(image_file, coder, not FLAGS.single_look)
    batch_results = sess.run(softmax_output, feed_dict={images:image_batch.eval()})
    output = batch_results[0]
    batch_sz = batch_results.shape[0]
    for i in range(1, batch_sz):
        output = output + batch_results[i]

    output /= batch_sz
    best = np.argmax(output)
    best_choice = (label_list[best], output[best])
    print('Guess @ 1 %s, prob = %.2f' % best_choice)

    #calculate face score
    score = scoreAge(output)

    nlabels = len(label_list)
    if nlabels > 2:
        output[best] = 0
        second_best = np.argmax(output)

        print('Guess @ 2 %s, prob = %.2f' % (label_list[second_best], output[second_best]))
    return label_list[best],score
```


其实就是输出概率最大值得标签，并从标签中拿到年龄段。
这里也献丑拿出我们的颜值分判别函数

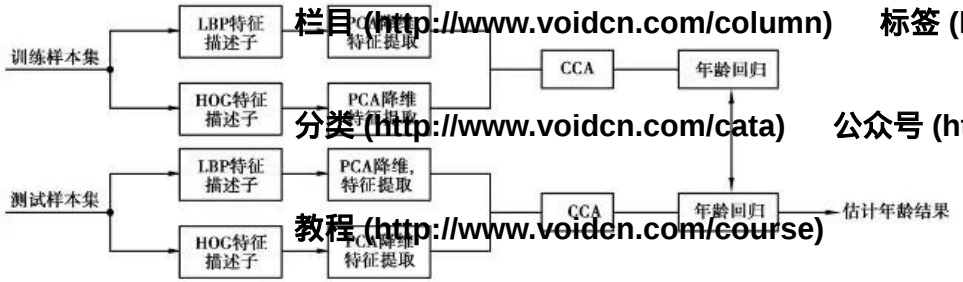
```
def scoreAge(prop_list):
    #['(0, 2)', '(4, 6)', '(8, 12)', '(15, 20)', '(25, 32)', '(38, 43)', '(48, 53)', '(60, 1
00)']
    #[90, 80, 80, 70, 60, 50, 40, 20]
    score_list = [100, 100, 95, 90, 80, 50, 40, 20]
    random_score_list = []
    finalScore = 0
    for score in score_list:
        randScore = score + random.random() * 10
        random_score_list.append(randScore)
    j = 0
    while(j < len(random_score_list)):
        prop = prop_list[j]
        score = random_score_list[j]
        finalScore = finalScore + score * prop
        j = j + 1
    return finalScore
```

不多说了，自己感受……

其他年龄识别算法

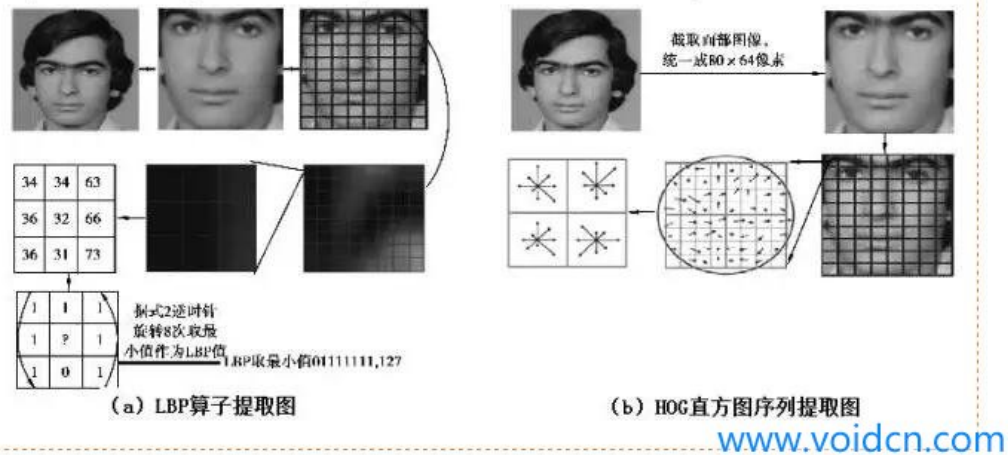
说到年龄估计的问题，定义并不明确。它既可以是分类问题，亦可是回归问题。如果将年龄分成几类，比如：少年、青年、中年和老年时，年龄估计就是分类问题；如果精确的估计具体年龄时，年龄估计就是回归问题。
说到底，年龄估计是一个比性别识别更为复杂的问题。原因在于：人的年龄特征在外表上很难准确地被观察出来，即使是人眼也很难准确地判断出一个人的年龄。再看人脸的年龄特征，它通常表现在皮肤纹理、皮肤颜色、光亮程度和皱纹纹理等方面，而这些因素通常与个人的遗传基因、生活习惯、性别和性格特征和工作环境等方面相关。所以说，我们很难用一个统一的模型去定义人脸图像的年龄。若想要较好地估出人的年龄层，则需要通过大量样本的学习，比如说年龄估计开始。

年龄估计大致分为预估和详细评估两个阶段。
预估阶段：
提取出照片中人脸的肌肤纹理特征，对年龄范围做一个大致的评估，得出一个特定的年龄段；
详细评估阶段：
通过支持向量机的方法，建立了对应于多个年龄段的多个模型分类器，并选择合适的模型进行匹配。这其中，以一项融合LBP和HOG特征的人脸年龄估计算法最为人们所熟知。



解析：

融合LBP和HOG特征的人脸年龄估计算法提取与年龄变化关系紧密的人脸的局部统计特征。LBP（局部二值化模式）特征和HOG（梯度直方图）特征，并用CCA（典型相关分析）的方法融合，最后通过SVR（支持向量机回归）的方法对人脸库进行训练和测试。



性别识别模型的训练过程

差不多

性别识别的预测过程

差不多

引申——怎么实现人脸颜值打分？

笔者提出融合多模型的一种方法，就是大杂烩了哈哈

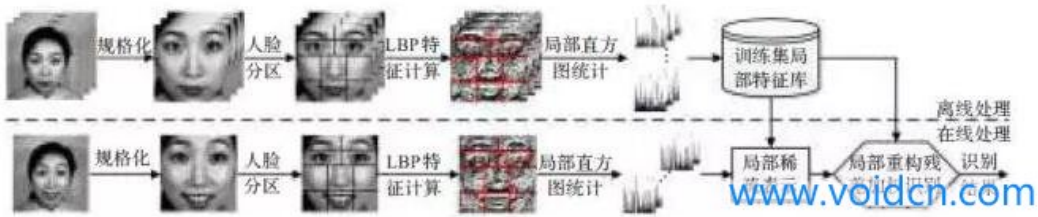
1. 可以尝试用深度学习，标注数据；
2. 表情识别，微笑颜值更高；

融合LBP 和局部稀疏表示的人脸表情识别算法包括：

首先，对规格化后的训练集人脸图像进行特征分区，对于没个人脸分区计算该区域的LBP特征，并采用直方图统计方法整合该区域特征向量，形成由特定人脸的局部特征组成的训练集局部特征库；

其次，对于测试人脸，同样进行人脸图像规格化、人脸分区、局部LBP特征计算和局部直方图统计操作；

最后，对于测试人脸的局部直方图统计特征，利用训练集特征库进行局部稀疏重构表示，并采用局部稀疏重构残差加权方法进行最终人脸表情分类识别。



3. 对称性判断，先验知识的加入，例如瓜子脸、锥子脸等分数高；
4. 光照效果，图片效果监测（采用AlexNet进行图片质量评价）；
5. 像明星的颜值更高，相似度模型。

Centos7环境安装步骤

1.安装Centos7
2.安装TensorFlow 0.12
下载安装包到Centos7上安装
3.安装openCV3.0
下载安装包到Centos7上编译安装
下载地址https://codeload.github.com/Itseez/opencv/zip/3.0.0
主要安装步骤参考http://www.cnblogs.com/asmer-stone/p/4592421.html
4.Python2.7自带Tornado

Tornado简介——一分钟搭建web服务的神器

Tornado 和现在的主流 Web 服务器框架（包括大多数 Python 的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。得利于其 非阻塞的方式和对 epoll 的运用，Tornado 每秒可以处理数以千计的连接，这意味着对于实时 Web 服务来说，Tornado 是一个理想的 Web 框架。

首先简单地介绍四中IO模型

常见的IO模型有四种（这四种模型在网络上也有很多很多的资料，为较少篇幅本片将这部分内容压缩一下）：

- 同步阻塞（Blocking IO）：最简单的一种IO模型，用户线程在进行IO操作的时候通常是个系统调用，用户线程会由用户空间进入内核空间，内核空间数据包准备好后会将数据拷贝到用户空间，这个时候线程在用户态继续执行。

- 同步非阻塞（Non-blocking IO）：同步非阻塞IO即在同步阻塞的基础之上将socket设置为NONBLOCK。这样用户线程在发起IO操作之后可以立即返回，但是用户线程需要不断轮询来请求数据。

- IO多路复用（IO Multiplexing）：即Reactor设计模式，多路复用模型从流程上和同步阻塞的区别不大，主要区别在于操作系统为用户提供了同时轮询多个IO句柄来查看是否有IO事件的接口（如select），这从根本上允许用户可以使用单个线程来管理多个IO句柄的问题。

- 异步IO（Asynchronous IO）：即Proactor设计模式。在异步IO模型中，用户不需要去轮询IO事件，然后才进行数据的读取，处理；在异步IO模型中，IO事件就绪的时候，内核会开启一个独立的内核线程去执行执行IO操作，实现真正的异步IO。这个时候用户线程可以直接读取内核线程准备好的数据。

这里就不得不对epoll、poll、select三种模型做一下简单介绍

select函数：改函数允许进程指示内核等待多个事件中的任何一个发生的时候或者在一定时间之后被唤醒，select有个致命的缺点即在多路复用中文件描述符的数量有限制，如果需要突破限制需要重新编译操作系统内核。

当用户进程调用了select，那么整个进程会被block，而同时，kernel会“监视”所有select负责的socket，当任何一个 socket中的数据准备好了，select就会返回。这个时候用户进程再调用read操作，将数据从kernel拷贝到用户进程。

poll函数：poll机制与select机制类似，区别是poll没有最大描述符限制。

epoll函数：epoll在linux2.6内核中被提出来，是之前的select和poll的增强版本。epoll也没有文件描述符数量限制，而且是用一个文件描述符来管理多个描述符。在性能上相比上面两种有了很大的优化。

JAVA对NIO的支持是从1.4版本开始的，是基于多路复用技术，而在linux操作系统方面多路复用技术有三种常用的机制：

select、poll和epoll

大家说哪里会用到呢，例如tomcat，例如我们的网关，要做一个高性能的网关程序，里面的门道还是比较多啊！

这里大家可以看看netty、mina的实现。

不是本文的主要目的，大家有兴趣的看文章吧

http://www.cnblogs.com/Anker/p/3265058.html

源码解析

程序园 (http://www.voidcn.com/)

整个流程很简单

栏目 (http://www.voidcn.com/column) **标签 (http://www.voidcn.com/tag)**

分类 (http://www.voidcn.com/cata) **公众号 (http://www.voidcn.com/public)**

教程 (http://www.voidcn.com/course)

```
# code: UTF-8
import tornado.ioloop
import tornado.web
import uuid
#import Image
import StringIO
import os
import commands
import guess
import json
import logging

#配置日志目录
logging.basicConfig(level=logging.DEBUG,
                format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %
(message)s',
                datefmt='%a, %d %b %Y %H:%M:%S',
                filename='facescore.log',
                filemode='w')

class PredictHandler(tornado.web.RequestHandler):

##网页Get访问
     def get(self):
            self.render("index.html")
##客户端App json soa服务
     def post(self):
        if self.request.files:
            file_name = "%s" % uuid.uuid1()
            print 'file_name',file_name
            file_raw = self.request.files["file"][0]["body"]
            usr_home = os.path.expanduser('~')
            file_name = usr_home+"/tensorflow/static/tmp/m_%s.jpg" % file_name
            fin = open(file_name,"w")
            print "success to open file"
            fin.write(file_raw)
```

```
            fin.close()
            print "use tensorflow"
            logging.info('start using tensorflow')
            ##人脸检测
            isman,file_name = guess.detectface(file_name)
            print 'detect face result, isman is %s, file name is %s '%(isman,file_name)
            feeds_json = {}
            if isman == 1:
                age,score = guess.guessage(file_name)
                gender = guess.guessGender(file_name)
                print 'guess age is ', age
                print 'gender is ',gender
                print 'score is ', score
                logging.info('filename is %s guess age is %s guess gender is %s guess score
 is %s '%(file_name,age,gender,score))
                dic = {}
                dic['ismankind'] = True
                dic['age'] = age
                dic['score'] = score
                dic['gender'] = gender
                feeds_json = json.dumps(dic)
            elif isman == 0 :
                logging.info('not mankind')
                dic = {}
                dic['ismankind'] = False
                feeds_json = json.dumps(dic)
            self.set_header('Content-Type', 'application/json; charset=UTF-8')
            self.write(feeds_json)
            self.finish()



application = tornado.web.Application([
    (r"/predict", PredictHandler),
])
```

```
if __name__ == "__main__":
    application.listen(8080)
    tornado.ioloop.IOLoop.instance().start()
```


首页测试html代码很简单

```
<html>
  <body>
    <form action="/practice/upload_formdata" method="post">
    <input name="file" type="file">
    <input type="submit" value="Submit">
    </form>
</body>
</html>
```

把这个放到公网服务器上，例如阿里云，然后后台启动
nohup python facescore.py 2>&1
就能正常访问了！
上传这个图片，还是喜欢看动作片啊！

返回json
{
“ismankind”: true,
“age”: “(38, 43)”,
“score”: 85.73050955477996,
“gender”: “M”
}

启动服务方法

nohup python facescore.py >/dev/null 2>&1 &
有点小技巧哦
/dev/null 表示空设备文件，相当于垃圾桶
0 表示stdin标准输入
1 表示stdout标准输出
2 表示stderr标准错误
2>1, 把标准错误stderr重定向到文件1中
2>&1,把标准错误stderr重定向到标准输出stdout
最后一个&后台执行
把标准输出和标准错误全重定向到空设备
nohup python facescore.py >/dev/null 2>&1

准确率较低的反思和改进方法

训练集要重构、改为中国人

要加入人脸抠图做训练，更容易发现特征

训练的时候，要发现最小的loss，及时停止保存model

数据是图像，就在已有的图像上进行随机的修改

TensorFlow debugger已经公布https://www.tensorflow.org/programmers_guide/debugger

请印出训练集的cost值和测试集上cost值的变化趋势，正常情况应该是训练集的cost值不断下降，最后趋于平缓，或者小范围震荡，测试集的cost值先下降，然后开始震荡或者慢慢上升。如果训练集cost值不下降，有可能是代码有bug，有可能是数据有问题（本身有问题，数据处理有问题等等），有可能是超参（网络大小，层数，学习率等）设置的不合理

优化算法可以更换其他试试

参考文献

- http://www.openu.ac.il/home/hassner/projects/cnn_agegender/
- https://github.com/GilLevi/AgeGenderDeepLearning
- https://cmusatyalab.github.io/openface/

- https://github.com/RiweiChen/DeepFace
- https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/
- http://blog.csdn.net/qq_14845119/article/details/51913171

相关文章

- 1. TensorFlow通过人脸识别年龄（2） (http://www.voidcn.com/article/p-pfjqbogs-oo.html)
- 2. TensorFlow通过人脸识别年龄（1） (http://www.voidcn.com/article/p-bgmvvasj-oo.html)
- 3. 基于CNN的人脸 性别、年龄识别 (http://www.voidcn.com/article/p-hnlntogv-ty.html)
- 4. 基于caffe的性别、年龄识别 (http://www.voidcn.com/article/p-gknctojw-pb.html)
- 5. 基于CNN的性别、年龄识别 (http://www.voidcn.com/article/p-yhfhnaht-bmd.html)
- 6. 人脸和性别识别（基于OpenCV） (http://www.voidcn.com/article/p-hqkjcbch-nm.html)
- 7. android人脸识别——HowOld测测你的年龄和性别 (http://www.voidcn.com/article/p-ebvjxoyi-uk.html)
- 8. 基于opencv的人脸性别识别 (http://www.voidcn.com/article/p-qkfwsweb-bde.html)
- 9. 基于CNN的性别、年龄识别及Demo实现 (http://www.voidcn.com/article/p-cmlyhwbo-xp.html)
- 10. 人脸属性识别算法 | 性别+种族+年龄+表情 (http://www.voidcn.com/article/p-uvnqvipt-oo.html)
- 更多相关文章... (http://www.voidcn.com/relative/p-qhajwjye-oo.html)

相关标签/搜索

性别-年龄识别 (http://www.voidcn.com/tag/%E6%80%A7%E5%88%AB-%E5%B9%B4%E9%BE%84%E8%AF%86%E5%88%AB) 性别年龄识别 (http://www.voidcn.com/tag/%E6%80%A7%E5%88%AB%E5%B9%B4%E9%BE%84%E8%AF%86%E5%88%AB) 年龄识别 (http://www.voidcn.com/tag/%E5%B9%B4%E9%BE%84%E8%AF%86%E5%88%AB) 人脸性别识别 (http://www.voidcn.com/tag/%E4%BA%BA%E8%84%B8%E6%80%A7%E5%88%AB%E8%AF%86%E5%88%AB) 人脸识别 (http://www.voidcn.com/tag/%E4%BA%BA%E8%84%B8%E8%AF%86%E5%88%AB) 人脸识别api (http://www.voidcn.com/tag/%E4%BA%BA%E8%84%B8%E8%AF%86%E5%88%ABapi) 人脸识别-Face+ (http://www.voidcn.com/tag/%E4%BA%BA%E8%84%B8%E8%AF%86%E5%88%AB-Face%2B) 人脸识别-dlib (http://www.voidcn.com/tag/%E4%BA%BA%E8%84%B8%E8%AF%86%E5%88%AB-dlib) 人脸识别 face++ (http://www.voidcn.com/tag/%E4%BA%BA%E8%84%B8%E8%AF%86%E5%88%AB+face%2B%2B) php人脸识别 (http://www.voidcn.com/tag/php%E4%BA%BA%E8%84%B8%E8%AF%86%E5%88%AB) 年龄识别 (http://www.voidcn.com/cata/6792656) 人脸识别 (http://www.voidcn.com/cata/1430233) 人脸识别 (http://www.voidcn.com/cata/5823251) 人脸识别 (http://www.voidcn.com/cata/1434512) 人脸识别 (http://www.voidcn.com/cata/1346053) 人脸识别 (http://www.voidcn.com/cata/1413345) 人脸识别

(http://www.voidcn.com/cata/2632095) 人脸识别 (http://www.voidcn.com/cata/1578803) 人脸识别
(http://www.voidcn.com/cata/1158853) 人脸识别 (http://www.voidcn.com/cata/5861985) Python
(http://www.voidcn.com/column/python) 从0到1实现基于Tornado和Tensorflow的人脸、年龄、性别识别
(http://www.voidcn.com/search/htavhy) opencv 人脸年龄识别 (http://www.voidcn.com/search/skbtkm) CNN 人脸年龄识别
(http://www.voidcn.com/search/urkmch) 基于Caffe的人脸识别实现 (http://www.voidcn.com/search/czudyb)
tensorflow实现人脸识别 (http://www.voidcn.com/search/cqubcc) 基于opencv实现人脸识别
(http://www.voidcn.com/search/wluorw) 基于openCV和LBP的人脸识别 (http://www.voidcn.com/search/djxuns)
tensorflow人脸识别 (http://www.voidcn.com/search/qmesgc) Tensorflow实现人脸识别 (http://www.voidcn.com/search/pnndyy)