Caffe学习教程

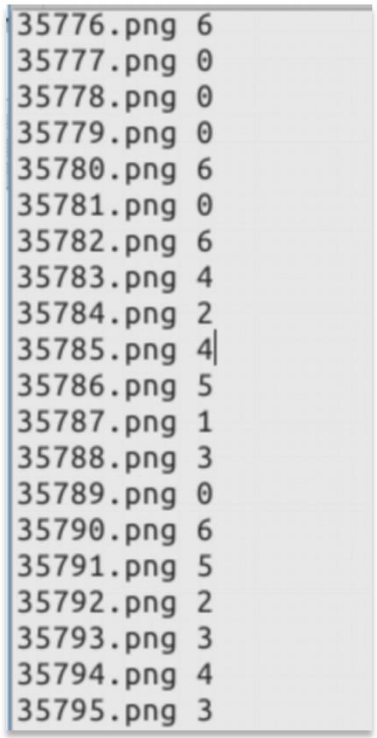
Contents

1、准备数据

- Caffe可以以下面的方式读取数据:
  - 从专用的数据库中读取（lmdb、leveldb)
  - 直接读取图片
  - 从内存中读取
  - 从HDF5文件中读取
  - 从滑动窗口中读取

1、准备数据

- 第一步：得到文件列表清单



Train.txt

1、准备数据

第二步：转换成lmdb

命令调用格式：
convert_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB_NAME

需要带四个参数：

FLAGS: 图片参数(可选)，用于调整图片大小和打乱顺序。

ROOTFOLDER/: 图片存放的绝对路径，从linux系统根目录开始

LISTFILE: 图片文件列表清单，一般为一个txt文件，一行一张图片

DB_NAME: 最终生成的db文件存放目录

例：convert_imageset /home/bnu/fer/train/ /home/bnu/fer/train.txt /home/bnu/fer/train_lmdb

1、准备数据

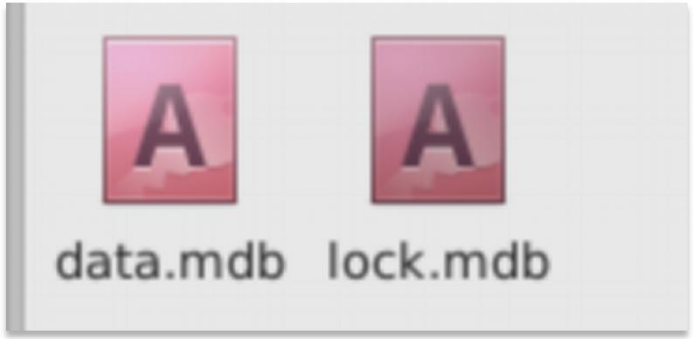
第二步：转换成lmdb

完整例子：

```
GLOG_logtostderr=1 $TOOLS
convert_imageset --shuffle \
--resize_height=256 \
--resize_width=256 \
/home/bnu/fer/train/ \
/home/bnu/fer/train.txt \
/home/bnu/fer/train_lmdb
```

打乱顺序
改变高度为256像素
改变宽度为256像素
图片存放的绝对路径
图片列表清单文件
生成的文件

转换成功后，会生成一个train_lmdb文件夹，里面有两个文件



更多参考：http://www.cnblogs.com/denny402/p/5082341.html

1、准备数据

第三步：计算均值

将训练集的均值计算出来，并保存为binaryporto文件，以便在训练时调用。

命令调用格式：

compute_image_mean param1 param2

Param1: lmdb文件夹
Param2: 保存文件路径及名称

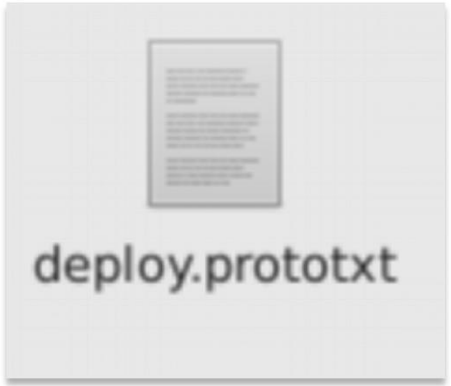
compute_image_mean /home/bnu/fer/train_lmdb /home/bnu/fer/mean.binaryproto

更多参考：http://www.cnblogs.com/denny402/p/5102328.html

2、构建网络结构

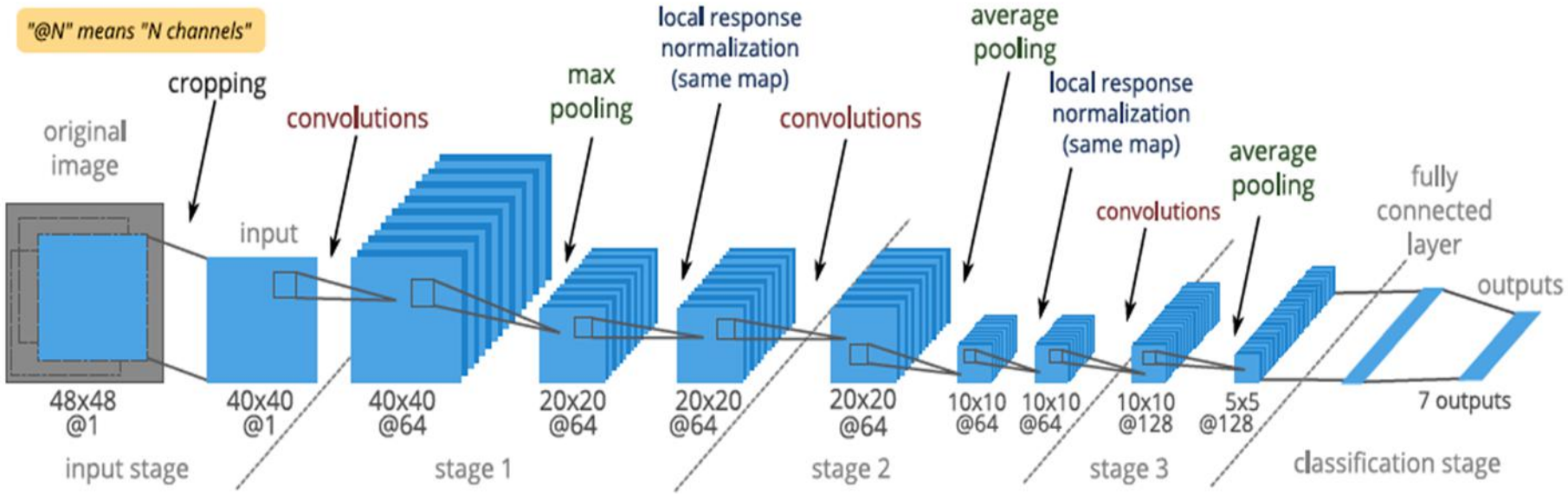
- 在运行的整个流程中，可以分为三个阶段：训练阶段、验证阶段和测试阶段。网络结构在不同的阶段是不同的，都存放在prototxt文件里面。为了方便，一般将训练阶段和验证阶段的网络结构放在一个文件里，测试阶段的网络结构单独放在一个文件里。

训练和验证阶段的网络结构文件



测试阶段的网络结构文件

2、构建网络结构


"@N" means "N channels"
original image
cropping
input
convolutions
max pooling
local response normalization (same map)
convolutions
average pooling
local response normalization (same map)
convolutions
average pooling
fully connected layer
outputs
48x48 @1
40x40 @1
40x40 @64
20x20 @64
20x20 @64
20x20 @64
10x10 @64
10x10 @64
10x10 @128
5x5 @128
7 outputs
input stage
stage 1
stage 2
stage 3
classification stage

2、构建网络结构（数据层)

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 40
    mean_file: "/home/xqh/fer/mean.binaryproto"
  }
  data_param {
    source: "/home/xqh/fer/fer/train_db"
    batch_size: 64
    backend: LMDB
  }
}
```

Name指定层的名称，不能有重复。
Type指定层的类型
自底向上，top用于指定向上传递的数据名称。数据层需要往上输出图片数据和标签数据。
Include用于指定该层属于训练阶段还是验证阶段

数据转换参数：
是否镜像
裁剪
均值文件

Lmdb数据源
批次大小
数据源格式

更多参考：http://www.cnblogs.com/denny402/p/5070928.html

2、构建网络结构（卷积层)

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1.0
    decay_mult: 1.0
  }
  param {
    lr_mult: 2.0
    decay_mult: 0.0
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0.0
    }
  }
}
```

bottom用于指定传入的数据名称，top用于该层的数据输出，bottom和top起到数据流动作用

lr_mult: 学习率的系数，最终的学习率是这个数乘以solver.prototxt配置文件中的base_lr。如果有两个lr_mult, 则第一个表示权值的学习率，第二个表示偏置项的学习率。一般偏置项的学习率是权值学习率的两倍。

bias 不更新

Num_output：卷积核的数量

Pad：边缘填充

Kernel_size:卷积核大小

Stride:滑动步长

$w_1=(w_0+2*pad-kernel\_size)/stride+1;$

$h_1=(h_0+2*pad-kernel\_size)/stride+1;$

Std是参数初始化成是高斯分布的标准差

Weight_filler:权值初始化，此处表示高斯随机初始化

Bias_filler: 偏置项初始化，此处表示初始化为常数0

更多参考：http://www.cnblogs.com/denny402/p/5071126.html

2、构建网络结构（激活层)

```
layer {
  name: "relu"
  type: "ReLU"
  bottom: “conv1"
  top: “relu1"
}
```

可选的激活函数有: ReLU, Sigmoid, TanH等

Sigmoid: $S(x) = \frac{1}{1+e^{-x}}$

ReLU: $f(x)=\max(x,0)$

TanH: $\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

更多参考：http://www.cnblogs.com/denny402/p/5072507.html

2、构建网络结构（池化层)

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: “relu1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
```

pool: 池化方法，默认为MAX。目前可用的方法有MAX, AVE, 或STOCHASTIC

pad: 和卷积层的pad的一样，进行边缘扩充。默认为0
stride: 池化的步长，默认为1。一般我们设置为2，也可以用stride_h和stride_w来设置。

w1=(w0+2*pad-kernel_size)/stride+1;
h1=(h0+2*pad-kernel_size)/stride+1;

更多参考：http://www.cnblogs.com/denny402/p/5071126.html

2、构建网络结构（全连接层)

```
layer {
  name: “fc7"
  type: "InnerProduct"
  bottom: "pool2"
  top: “fc7"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

全连接层是一种特殊的卷积层，只是一种是全连接，一种是局部连接。因此参数基本是一致的。

神经网络（NN）或多层感知机(NLP)就只有全连接层

更多参考：http://www.cnblogs.com/denny402/p/5072746.html

2、构建网络结构（其它层)

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc7"
  bottom: "label"
  top: "loss"
}
```

Loss层：输出残差

```
layers {
  bottom: “fc7"
  top: "prob"
  name: "prob"
  type: “Softmax“
 include {
    phase: TEST
  }
}
```

Softmax层：输出属于某类的概率,一般用于验证和测试阶段，这里的TEST表示验证阶段

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: “fc7"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
```

输出分类（预测）精确度，只有验证或测试阶段才有

```
layer {
  name: "drop7"
  type: "Dropout"
  bottom: "fc7"
  top: "fc7"
  dropout_param {
    dropout_ratio: 0.5
  }
}
```

Dropout是一个防止过拟合的trick。可以随机让网络某些隐含层节点的权重不工作

2、构建网络结构（deploy)

Deploy.prototxt文件用于测试阶段，测试数据没有标签值，因此数据输入层与其它两个阶段不同。

```
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 42
  dim: 42
}
```

3、配置参数（solver.prototxt)

net: “/home/bnu/fer/train_val.prototxt"	网络结构文件
test_iter: 28	验证迭代次数
test_interval: 5000	验证间隔
base_lr: 0.005	基础学习率
display: 50	屏幕显示间隔
max_iter: 300000	最大训练次数
lr_policy: "step"	学习率变化规则
gamma: 0.1	学习率变化系数
stepsize: 8891	学习率变化系数
momentum: 0.9	动量
weight_decay: 0.0005	权值衰减系数
snapshot: 3000	Model保存间隔
snapshot_prefix: "snapshot"	Model名字前缀
solver_mode: GPU	硬件配置
solver_type: SGD	优化方法

Fer库有1280000张训练图片，7168张验证图片

设置batch_size=256, 则迭代1280000/256 =5000次才完整训练完一次所有图片（1 epoch)，迭代7168/256 =28次才完整验证完一次所有图片。

如果我们想训练60 epoch，则最大训练次数为60* 5000=300000

更多参考：http://www.cnblogs.com/denny402/p/5074049.html

3、配置参数（solver.prototxt)

lr_policy可以设置为下面这些值，相应的学习率的计算为：

- Fixed：保持base_lr不变.

- Step：如果设置为step,则还需要设置一个stepsize，返回 base_lr * gamma ^ (floor(iter / stepsize)),其中iter表示当前的迭代次数

- Exp：返回base_lr * gamma ^ iter， iter为当前迭代次数

- Inv：如果设置为inv,还需要设置一个power, 返回base_lr * (1 + gamma * iter) ^ (- power)

- Multistep：如果设置为multistep,则还需要设置一个stepvalue。这个参数和step很相似，step是均匀等间隔变化，而multistep则是根据stepvalue值变化

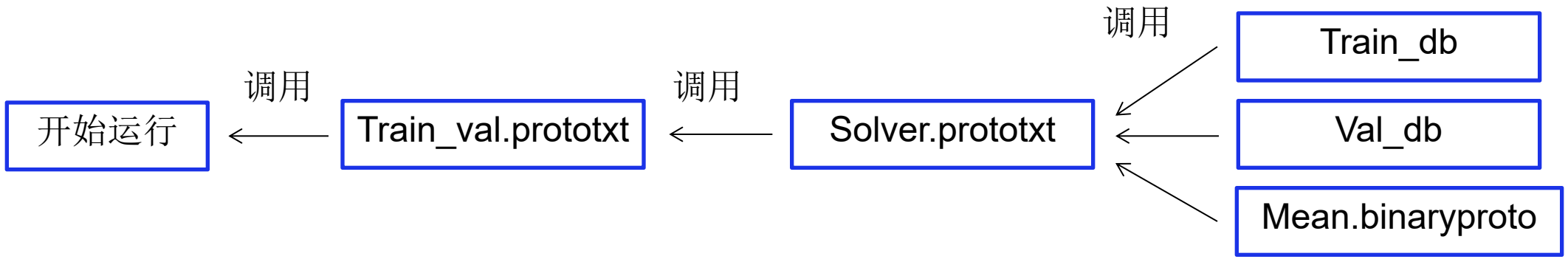
- Poly：学习率进行多项式误差, 返回 base_lr (1 - iter/max_iter) ^ (power)

- Sigmoid：学习率进行sigmod衰减，返回 base_lr (1/(1 + exp(-gamma * (iter - stepsize))))

更多参考：http://www.cnblogs.com/denny402/p/5074049.html

4、训练模型


train_db
val_db
deploy.prototxt
train_val.prototxt
mean.binaryproto


调用
调用
调用
开始运行
Train_val.prototxt
Solver.prototxt
Train_db
Val_db
Mean.binaryproto

4、训练模型

caffe程序的命令行执行格式如下：

caffe <command> <args>

其中的<command>有这样四种：

- train
- test
- device_query
- Time

对应的功能为：

train----训练或finetune模型（model),

test-----测试模型

device_query---显示gpu信息

time-----显示程序执行时间

其中的<args>参数有：

-solver	必须，训练配置文件
-gpu	可选，指定某块gpu
-snapshot	可选，从快照中恢复训练
-weights	可选，预训练好的caffemodel
-iteration	可选，迭代次数
-model	可选，网络结构
-sighup_effect	意外中止时的操作
-sigint_effect	人工中止时的操作

4、训练模型

例：

build/tools/caffe train -solver /home/bnu/fer/solver.prototxt

build/tools/caffe train –solver=/home/bnu/fer/solver.prototxt

更多参考：http://www.cnblogs.com/denny402/p/5076285.html

5、测试模型

经过前面几步操作，我们已经训练好了一个caffemodel模型，并生成了一个deploy.prototxt文件，我们就利用这两个文件来对一个新的图片进行分类预测。

将所有的类别名称，写入一个txt文件如（如class.txt)，每行一个类别，再执行下列命令：

命令行：

```
build/examples/cpp_classification/classification.bin \
/home/bnu/fer/deploy.prototxt \
/home/bnu/fer/fer.caffemodel \
/home/bnu/fer/mean.binaryproto \
/home/bnu/fer/class.txt \
/home/bnu/val/005.jpg
```

第一个参数：depoly配置文件
第二个参数：caffemodel文件
第三个参数：均值文件
第四个参数：类别名称文件
第五个参数：要分类的图片

更多参考：http://www.cnblogs.com/denny402/p/5111018.html

深度学习框架总结

库名	主语言	从语言	速度	灵活性	上手难易	开发者
TensorFlow	C++	cuda/python/Matlab/Ruby/R	中等	好	难	Google
Caffe	C++	cuda/python/Matlab	快	一般	中等	贾扬清
PyTorc	python	C/C++	中等	好	中等	FaceBook
MXNet	C++	cuda/R/julia	快	好	中等	李沐和陈天奇等
Torch	lua	C/cuda	快	好	中等	Facebook
Theano	python	C++/cuda	中等	好	易	蒙特利尔理工学院

Thank You !