**一、下载model参数**

**可以直接在浏览器里输入地址下载，也可以运行脚本文件下载。下载地址为：**[**http://dl.caffe.berkeleyvision.org/bvlc\_reference\_caffenet.caffemodel**](http://dl.caffe.berkeleyvision.org/bvlc_reference_caffenet.caffemodel)

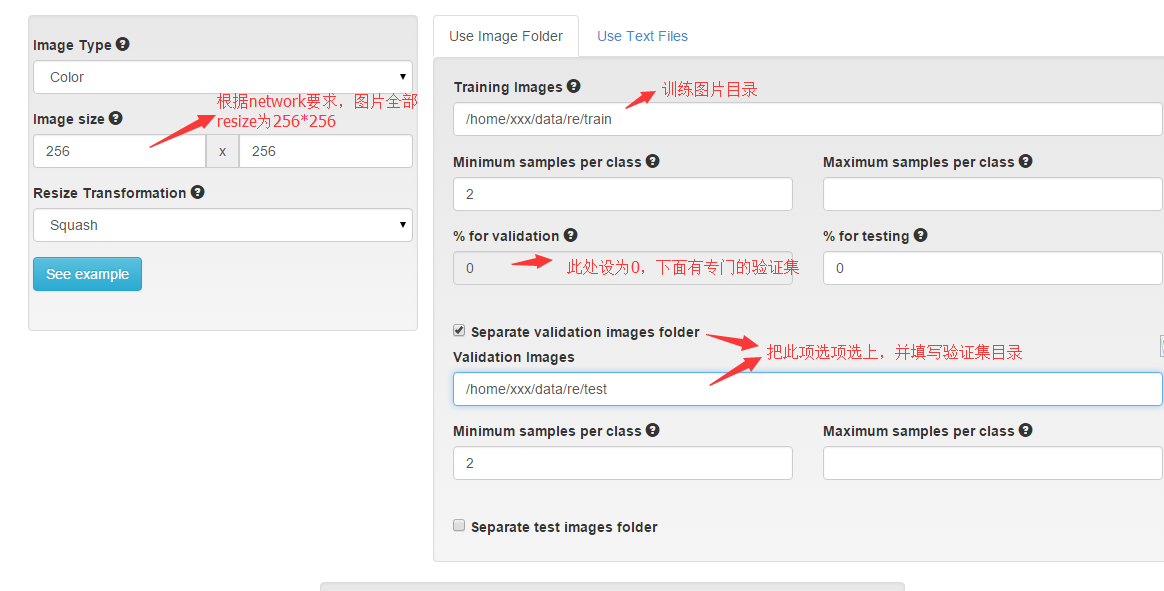
**文件名称为：bvlc\_reference\_caffenet.caffemodel，文件大小为230M左右，为了代码的统一，将这个caffemodel文件下载到caffe根目录下的 models/bvlc\_reference\_caffenet/ 文件夹下面。也可以运行脚本文件进行下载：**

**# sudo ./scripts/download\_model\_binary.py models/bvlc\_reference\_caffenet**

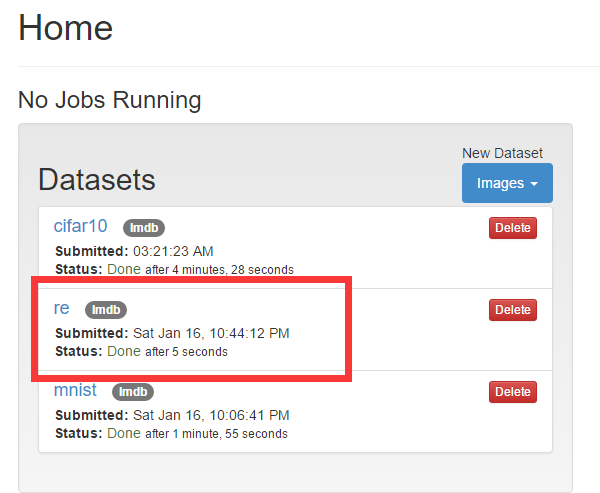
**二、准备数据**

**将训练数据放在一个文件夹内。比如我在当前用户根目录下创建了一个data文件夹，专门用来存放数据，因此我的训练图片路径为：/home/xxx/data/re/train**

**打开浏览器，运行digits，新建一个classification dataset,设置如下图：**

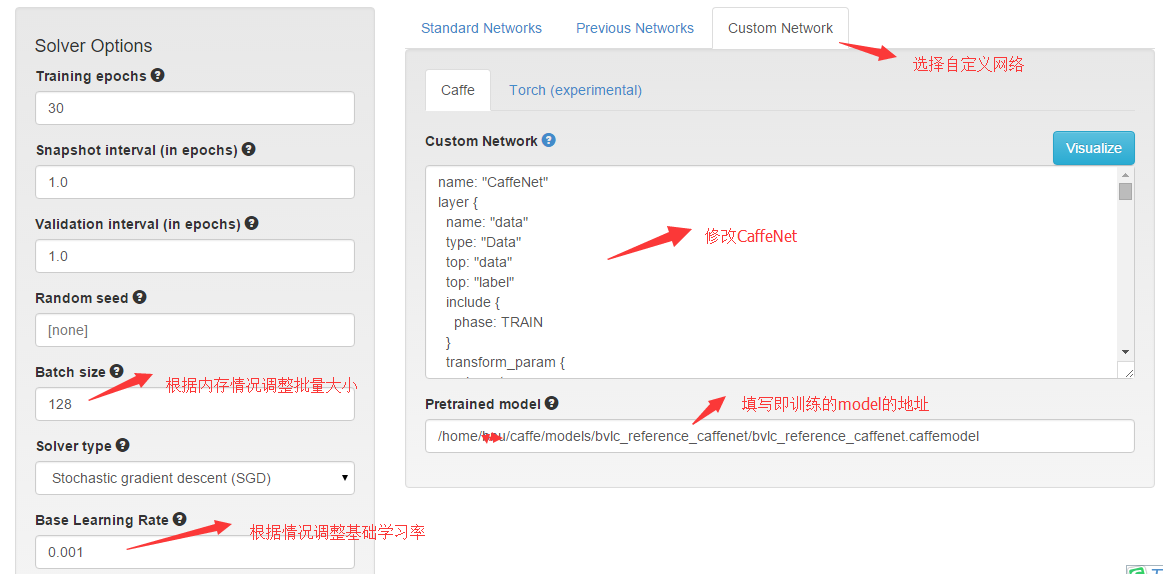
****

**下面图片格式选为jpg, 为dataset取一个名字，就开始转换吧。结果如图：**

****

**三、设置model**

**回到digits根目录，新建一个classification model， 选中你的dataset, 开始设置最重要的network.**

****

**caffenet的网络配置文件，放在 caffe/models/bvlc\_reference\_caffenet/ 这个文件夹里面，名字叫train\_val.prototxt。打开这个文件，将里面的内容复制到上图的Custom Network文本框里，然后进行修改，主要修改这几个地方：**

**1、修改train阶段的data层为：**

**layer {**

**name: "data"**

**type: "Data"**

**top: "data"**

**top: "label"**

**include {**

**phase: TRAIN**

**}**

**transform\_param {**

**mirror: true**

**crop\_size: 227**

**}**

**}**

**即把均值文件（mean\_file)、数据源文件(source)、批次大小(batch\_size)和数据源格式（backend)这四项都删除了。因为这四项系统会根据dataset和页面左边“solver options"的设置自动生成。如果想用原始数据训练，可以不用crop\_size，即图像数据不会crop,按照原始图像大小训练。**

**2、修改test阶段的data层：和上面一样，也是删除那些项。**

**layer {**

**name: "data"**

**type: "Data"**

**top: "data"**

**top: "label"**

**include {**

**phase: TEST**

**}**

**transform\_param {**

**mirror: false**

**crop\_size: 227**

**}**

**}**

**3、修改最后一个全连接层（fc8)：**

**layer {**

**name: "fc8-re" #原来为"fc8"**

**type: "InnerProduct"**

**bottom: "fc7"**

**top: "fc8"**

**param {**

**lr\_mult: 1.0**

**decay\_mult: 1.0**

**}**

**param {**

**lr\_mult: 2.0**

**decay\_mult: 0.0**

**}**

**inner\_product\_param {**

**num\_output: 5 #原来为"1000"**

**weight\_filler {**

**type: "gaussian"**

**std: 0.01**

**}**

**bias\_filler {**

**type: "constant"**

**value: 0.0**

**}**

**}**

**}**

**看注释的地方，就只有两个地方修改，其它不变。**

**设置好后，就可以开始微调了(fine tuning).**

**训练结果就是一个新的model，可以用来单张图片和多张图片测试。在此，将别人训练好的model用到我们自己的图片分类上，整个微调过程就是这样了。如果你不用digits，而直接用命令操作，那就更简单，只需要修改一个train\_val.prototxt的配置文件就可以了，其它都是一样的操作。**

**【注意】新版digits的网络结构是针对所有网络的，即包括的训练的网络结构，测试的网络结构和验证的网络结构，即在一个.prototxt 中包含了train/val/deploy 所有的结构。**

**如果使用新版digits，除了上面数据层和最后一个全连接层的改动外，还有以下3处：**

**（1）修改accuracy层，删除原来phase: TEST修改为stage: "val"，下图的-表示删除，+表示增加，后面的均是这样表示。**

|  |
| --- |
| **layer {**  **name: "accuracy"** |
| **type: "Accuracy"** |
| **bottom: "output"** |
| **bottom: "label"** |
| **top: "accuracy"** |
| **- include {** |
| **- phase: TEST** |
| **- }** |
| **+ include { stage: "val" }** |
| **+}** |

**（2）修改loss层，增加exclude { stage: "deploy" }，表示loss只在训练和验证中计算，测试时不计算。**

|  |
| --- |
| **layer {** |
| **name: "loss"** | |
| **type: "SoftmaxWithLoss"** | |
| **bottom: "output"** | |
| **bottom: "label"** | |
| **top: "loss"** | |
| **+ exclude { stage: "deploy" }**  **+}** | |

**（3）增加softmax层，该层不在训练和验证中计算，只在测试时计算。**

|  |
| --- |
| **+ layer {** |
| **+ name: "softmax"** | |
| **+ type: "Softmax"** | |
| **+ bottom: "output"** | |
| **+ top: "softmax"** | |
| **+ include { stage: "deploy" }**  **+}** | |