Prepare the Dataset

```
cd $CAFFE_ROOT
./data/cifar10/get_cifar10.sh
```

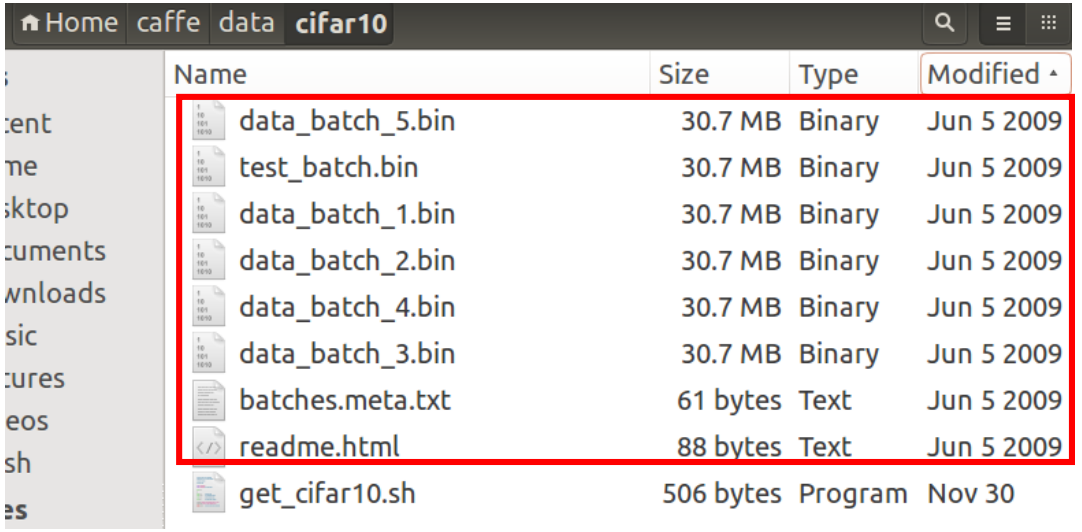
```
chg0901@ubuntu:~/caffe$ ./data/cifar10/get_cifar10.sh
Downloading...
--2016-12-01 11:54:22--  http://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz
Resolving www.cs.toronto.edu (www.cs.toronto.edu)... 128.100.3.30
Connecting to www.cs.toronto.edu (www.cs.toronto.edu)|128.100.3.30|:80... connec
ted.
HTTP request sent, awaiting response... 200 OK
Length: 170052171 (162M) [application/x-gzip]
Saving to: ‘cifar-10-binary.tar.gz.1’

48% [=================>                   ] 82,374,356  3.21MB/s   eta 31s
```

```
100%[======================================>] 170,052,171 4.50MB/s   in 50s

2016-12-01 11:56:51 (3.26 MB/s) - ‘cifar-10-binary.tar.gz’ saved [170052171/1700
52171]

Unzipping...
Done.
```

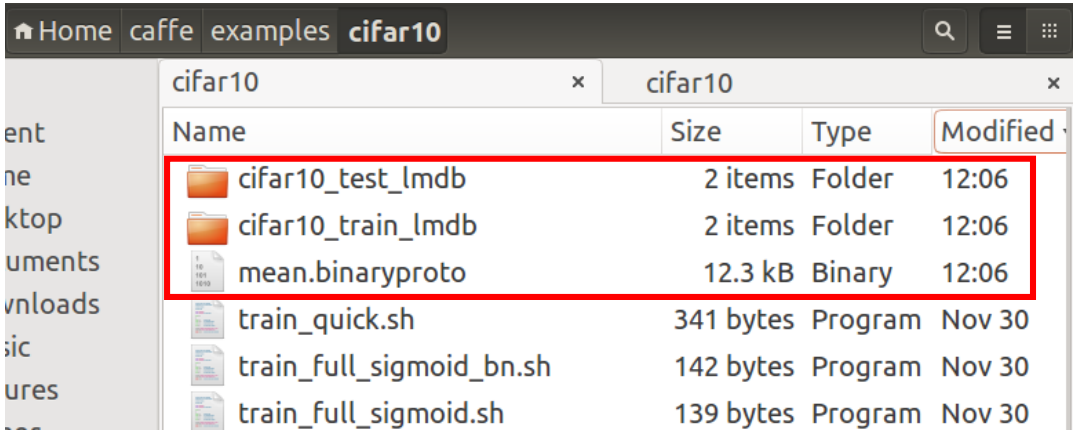


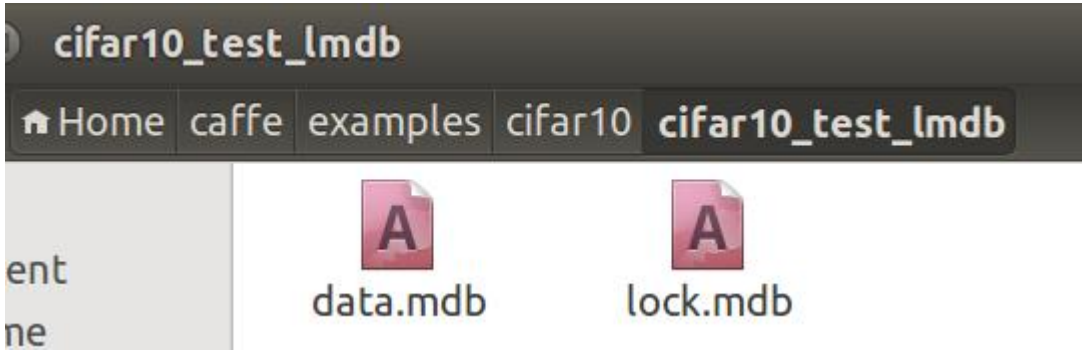
PS：Use wget and gunzip in the script file

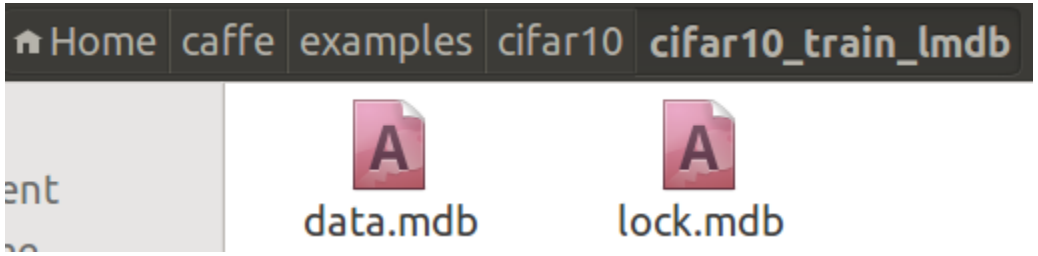
Prepare the Dataset

./examples/cifar10/create_cifar10.sh

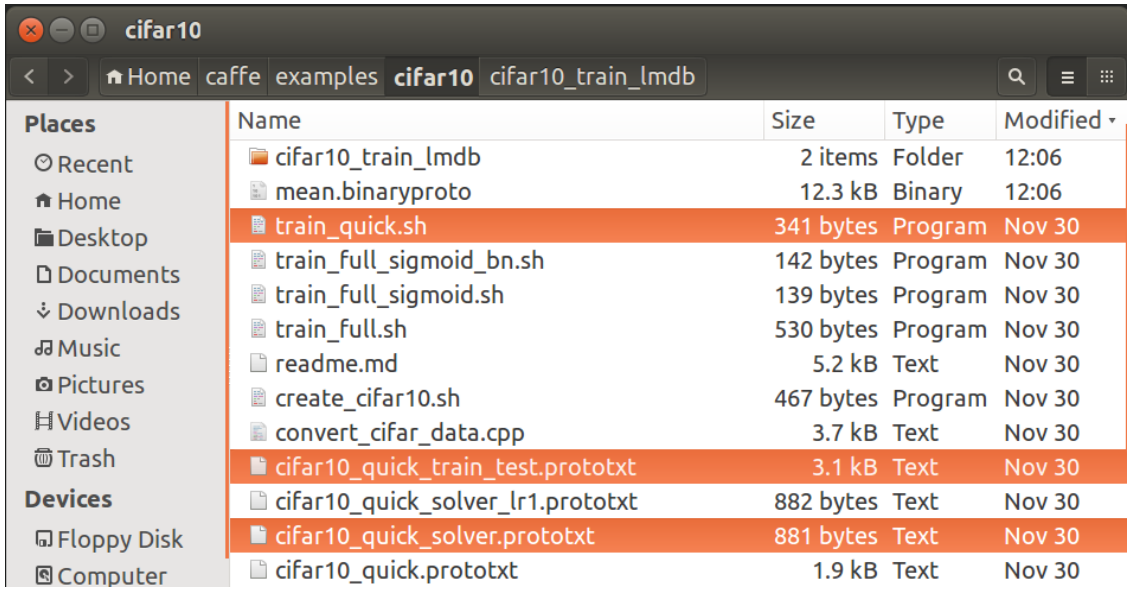
```
Creating lmdb...
I1201 12:06:36.980993  3240 db_lmdb.cpp:35] Opened lmdb examples/cifar10/cifar10
_train_lmdb
I1201 12:06:36.981171  3240 convert_cifar_data.cpp:52] Writing Training data
I1201 12:06:36.981179  3240 convert_cifar_data.cpp:55] Training Batch 1
I1201 12:06:37.014691  3240 convert_cifar_data.cpp:55] Training Batch 2
I1201 12:06:37.045920  3240 convert_cifar_data.cpp:55] Training Batch 3
I1201 12:06:37.095429  3240 convert_cifar_data.cpp:55] Training Batch 4
I1201 12:06:37.128257  3240 convert_cifar_data.cpp:55] Training Batch 5
I1201 12:06:39.879438  3240 convert_cifar_data.cpp:73] Writing Testing data
I1201 12:06:39.879758  3240 db_lmdb.cpp:35] Opened lmdb examples/cifar10/cifar10
_test_lmdb
Computing image mean...
Done.
chg0901@ubuntu:~/caffe$ ^C
```







The Model



train_quick.sh ×

```
#!/usr/bin/env sh
set -e

TOOLS=./build/tools

$TOOLS/caffe train \
  --solver=examples/cifar10/cifar10_quick_solver.prototxt $@

# reduce learning rate by factor of 10 after 8 epochs
$TOOLS/caffe train \
  --solver=examples/cifar10/cifar10_quick_solver_lr1.prototxt \
  --snapshot=examples/cifar10/cifar10_quick_iter_4000.solverstate.h5 $@
```

cifar10_quick_train_test.prototxt (~/caffe/examples/cifar10) - gedit

Open Save Undo

cifar10_quick_train_test.prototxt ×

```
name: "CIFAR10_quick"
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  data_param {
    source: "examples/cifar10/cifar10_train_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  data_param {
    source: "examples/cifar10/cifar10_test_lmdb"
    batch_size: 100
    backend: LMDB
```

The Model

cifar10_quick_solver.prototxt ×

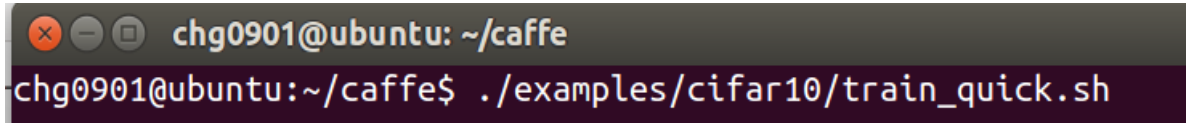
```
# reduce the learning rate after 8 epochs (4000 iters) by a factor of 10

# The train/test net protocol buffer definition
net: "examples/cifar10/cifar10_quick_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.001
momentum: 0.9
weight_decay: 0.004
# The learning rate policy
lr_policy: "fixed"
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 4000
# snapshot intermediate results
snapshot: 4000
snapshot_format: HDF5
snapshot_prefix: "examples/cifar10/cifar10_quick"
# solver mode: CPU or GPU
solver_mode: GPU
```

without GPU then change it as CPU

Training and Testing the “Quick” Model

./examples/cifar10/train_quick.sh




```
I1201 13:33:11.354599  3990 layer_factory.hpp:77] Creating layer cifar
I1201 13:33:11.374011  3990 net.cpp:100] Creating Layer cifar
I1201 13:33:11.387137  3990 net.cpp:408] cifar -> data
I1201 13:33:11.387363  3990 net.cpp:408] cifar -> label
I1201 13:33:11.387559  3990 data_transformer.cpp:25] Loading mean file from: exa
mples/cifar10/mean.binaryproto
I1201 13:33:11.397881  3993 db_lmdb.cpp:35] Opened lmdb examples/cifar10/cifar10
_train_lmdb
I1201 13:33:11.398475  3990 data_layer.cpp:41] output data size: 100,3,32,32
I1201 13:33:11.453248  3990 net.cpp:150] Setting up cifar
I1201 13:33:11.453389  3990 net.cpp:157] Top shape: 100 3 32 32 (307200)
I1201 13:33:11.453438  3990 net.cpp:157] Top shape: 100 (100)
I1201 13:33:11.453443  3990 net.cpp:165] Memory required for data: 1229200
I1201 13:33:11.453451  3990 layer_factory.hpp:77] Creating layer conv1
I1201 13:33:11.453480  3990 net.cpp:100] Creating Layer conv1
I1201 13:33:11.453498  3990 net.cpp:434] conv1 <- data
I1201 13:33:11.453531  3990 net.cpp:408] conv1 -> conv1
I1201 13:33:11.459303  3990 net.cpp:150] Setting up conv1
I1201 13:33:11.459343  3990 net.cpp:157] Top shape: 100 32 32 32 (3276800)
```

It’s the details about each layer, its connections and its output shape

Training and Testing the “Quick” Model

```
I1201 13:49:36.128219  4174 net.cpp:283] Network initialization done.
I1201 13:49:36.128301  4174 solver.cpp:60] Solver scaffolding done.
I1201 13:49:36.128336  4174 caffe.cpp:251] Starting Optimization
I1201 13:49:36.128351  4174 solver.cpp:279] Solving CIFAR10_quick
I1201 13:49:36.128355  4174 solver.cpp:280] Learning Rate Policy: fixed
I1201 13:49:36.128662  4174 solver.cpp:337] Iteration 0, Testing net (#0)
I1201 13:50:01.812454  4174 solver.cpp:404]     Test net output #0: accuracy = 0.087
I1201 13:50:01.812531  4174 solver.cpp:404]     Test net output #1: loss = 2.30256 (* 1 = 2.30256 loss
)
I1201 13:50:02.441434  4174 solver.cpp:228] Iteration 0, loss = 2.30204
```

Based on the solver setting

print the training loss function every 100 iterations

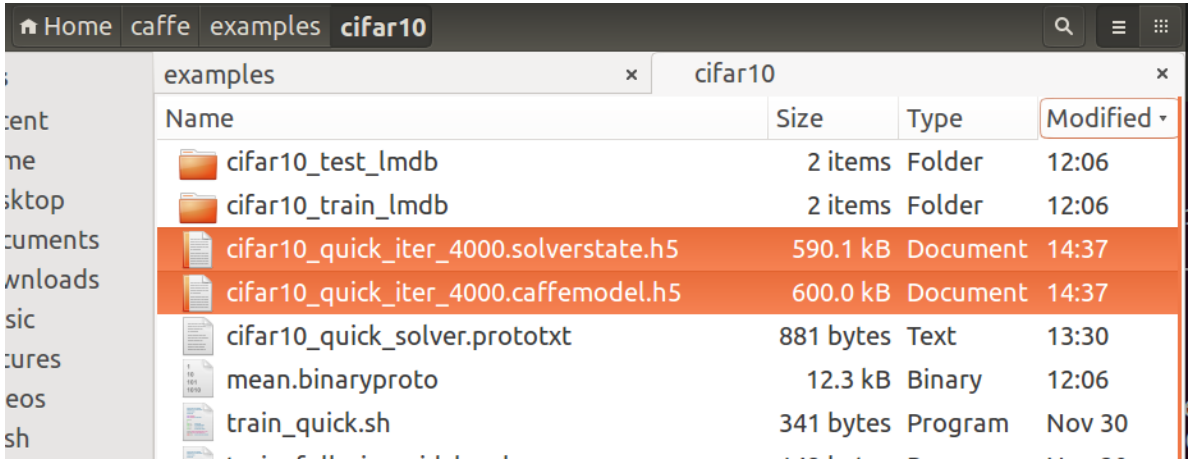
test the network every 500 iterations

```
I1201 13:54:21.006883  4174 solver.cpp:228] Iteration 400, loss = 1.23805
I1201 13:54:21.008085  4174 solver.cpp:244]     Train net output #0: loss = 1.23805 (* 1 = 1.23805 los
s)
I1201 13:54:21.008096  4174 sgd_solver.cpp:106] Iteration 400, lr = 0.001
I1201 13:55:23.867566  4174 solver.cpp:337] Iteration 500, Testing net (#0)
I1201 13:55:49.855676  4174 solver.cpp:404]     Test net output #0: accuracy = 0.5541
I1201 13:55:49.855830  4174 solver.cpp:404]     Test net output #1: loss = 1.28027 (* 1 = 1.28027 loss
)
```

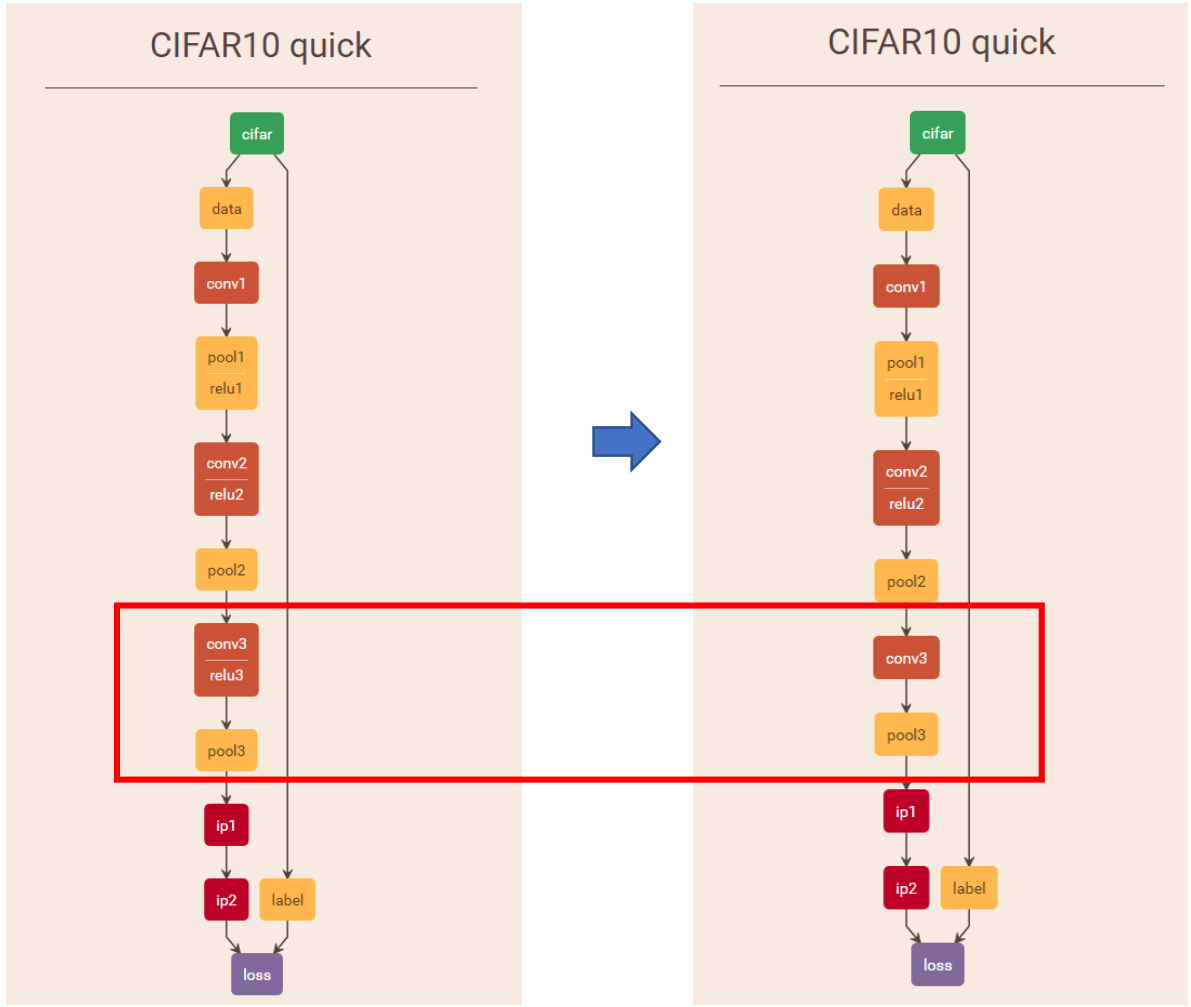
Result of Train

```
I1201 14:37:06.223274  4174 solver.cpp:464] Snapshotting to HDF5 file examples/cifar10/cifar10_quick_i
ter_4000.caffemodel.h5
I1201 14:37:06.227216  4174 sgd_solver.cpp:283] Snapshotting solver state to HDF5 file examples/cifar1
0/cifar10_quick_iter_4000.solverstate.h5
I1201 14:37:06.506415  4174 solver.cpp:317] Iteration 4000, loss = 0.669315
I1201 14:37:06.506522  4174 solver.cpp:337] Iteration 4000, Testing net (#0)
I1201 14:37:31.641747  4174 solver.cpp:404]     Test net output #0: accuracy = 0.7196
I1201 14:37:31.641837  4174 solver.cpp:404]     Test net output #1: loss = 0.846161 (* 1 = 0.846161 lo
ss)
I1201 14:37:31.641862  4174 solver.cpp:322] Optimization Done.
I1201 14:37:31.641866  4174 caffe.cpp:254] Optimization Done.
```

The model parameters are stored in binary protobuf format in



Make your own Net



A web-based tool for visualizing neural network architectures
http://ethereon.github.io/netscope/quickstart.html

Just change the "**cifar10_quick_train_test.prototxt**"

Change the 'bottom' and 'top' of the layer you want to add （define the required parameters and choose the optional parameters） or delete （comment by #）

```
122 layer {
123   name: "conv3"
124   type: "Convolution"
125   bottom: "pool2"
126   top: "conv3"
127   param {
128     lr_mult: 1
129   }
130   param {
131     lr_mult: 2
132   }
133   convolution_param {
134     num_output: 64
135     pad: 2
136     kernel_size: 5
137     stride: 1
138     weight_filler {
139       type: "gaussian"
140       std: 0.01
141     }
142     bias_filler {
143       type: "constant"
144     }
145   }
146 }
```

```
147 #layer {
148 # name: "relu3"
149 # type: "ReLU"
150 # bottom: "conv3"
151 # top: "conv3"
152 #}
153 layer {
154   name: "pool3"
155   type: "Pooling"
156   bottom: "conv3"
157   top: "pool3"
158   pooling_param {
159     pool: AVE
160     kernel_size: 3
161     stride: 2
162   }
163 }
```

Then train and test again, look at what will happen.