Caffe

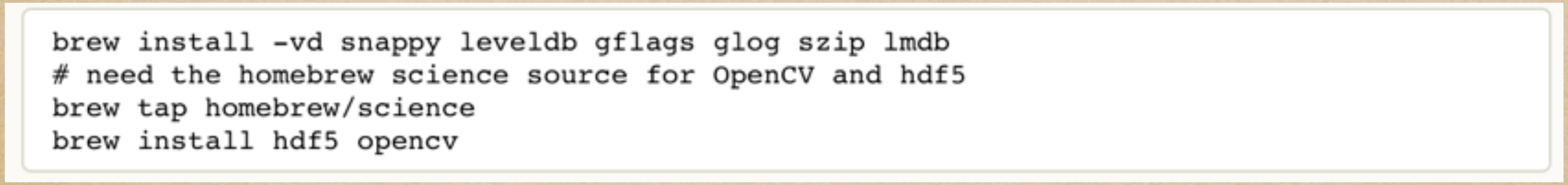
Tian Huang
2016011379
April 11,2017
t-huang16@mails.tsinghua.edu.cn

Installation

- Mac OS 10.12 Sierra
- Homebrew (package manager)
- CUDA 7 (useless without GPU)
- Anaconda Python
- Modify environment variables

Installation

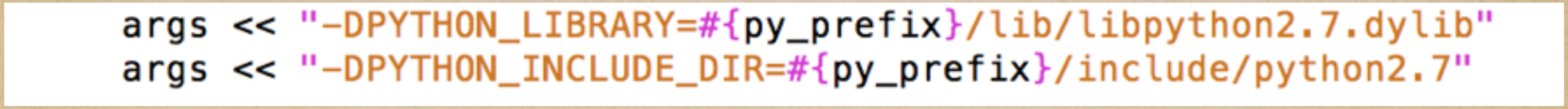
- install OpenCV & hdf5



```
brew install -vd snappy leveldb gflags glog szip lmdb
# need the homebrew science source for OpenCV and hdf5
brew tap homebrew/science
brew install hdf5 opencv
```

- if using Anaconda python, hdf5 can be skipped.

- brew edit opencv



```
    args << "-DPYTHON_LIBRARY=#{py_prefix}/lib/libpython2.7.dylib"
    args << "-DPYTHON_INCLUDE_DIR=#{py_prefix}/include/python2.7"
```

Installation

- with Python
- brew install --build-from-source --with-python -vd protobuf
- brew install --build-from-source -vd boost boost-python

- without Python
- brew install protobuf boost

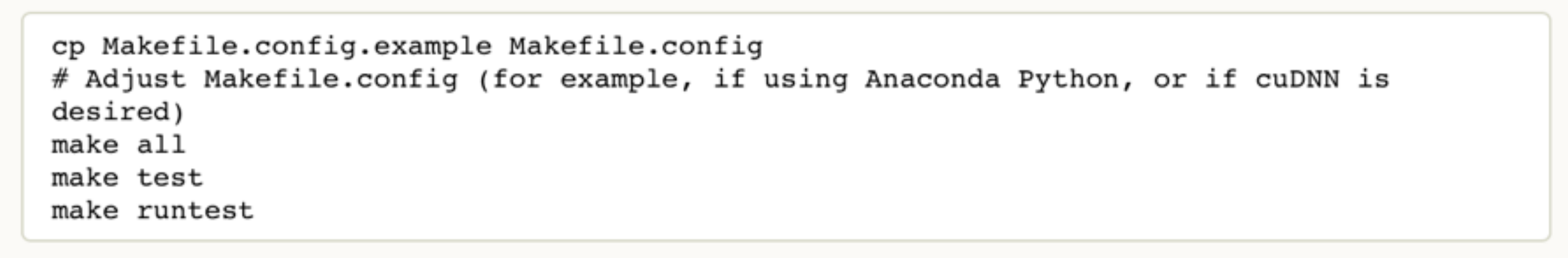
Installation

- BLAS (or MKL)

- Python (without Anaconda)

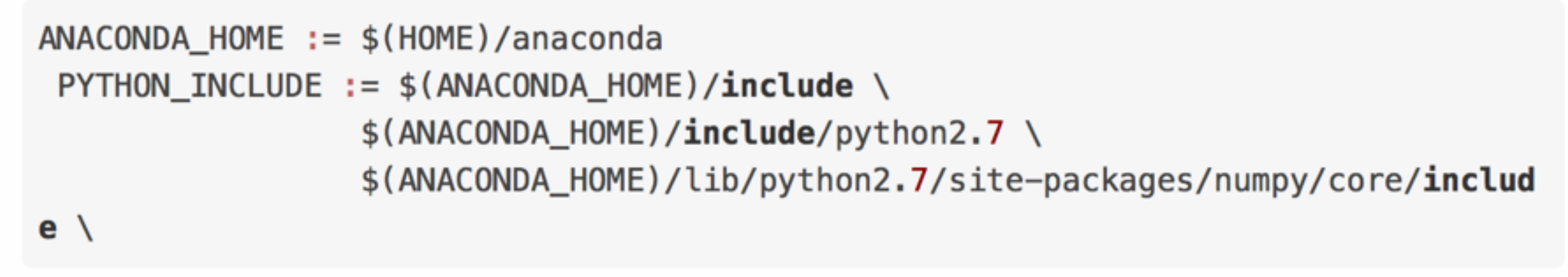
- `for req in $(cat requirements.txt); do pip install $req; done`

Compilation



```
cp Makefile.config.example Makefile.config
# Adjust Makefile.config (for example, if using Anaconda Python, or if cuDNN is
desired)
make all
make test
make runtest
```

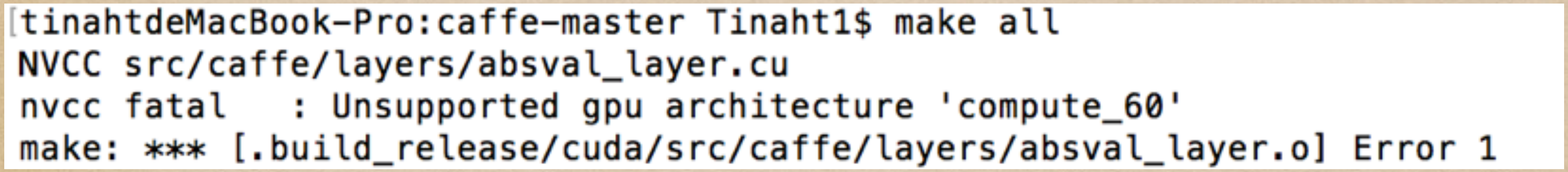
- Notice:
- Without GPU:
- uncomment CPU_ONLY := 1 in Makefile.config



```
ANACONDA_HOME := $(HOME)/anaconda
 PYTHON_INCLUDE := $(ANACONDA_HOME)/include \
                $(ANACONDA_HOME)/include/python2.7 \
                $(ANACONDA_HOME)/lib/python2.7/site-packages/numpy/core/includ
e \
```

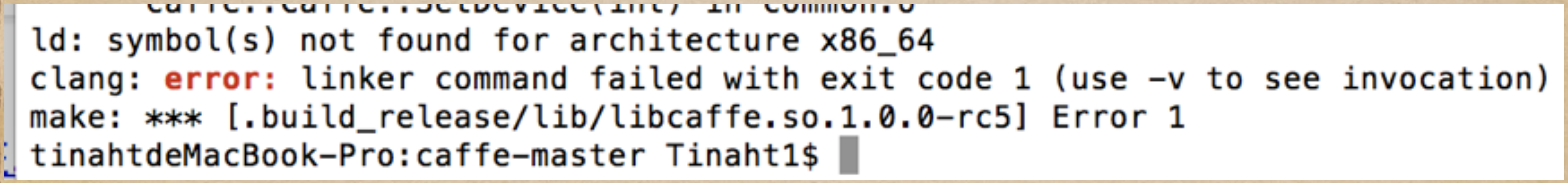
Compilation

- if you forget it



```
[tinahtdeMacBook-Pro:caffe-master Tinaht1$ make all
NVCC src/caffe/layers/absval_layer.cu
nvcc fatal   : Unsupported gpu architecture 'compute_60'
make: *** [.build_release/cuda/src/caffe/layers/absval_layer.o] Error 1
```

- when you try to modify



```
    caffe::caffe::SetDevice(int) in common.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
make: *** [.build_release/lib/libcaffe.so.1.0.0-rc5] Error 1
tinahtdeMacBook-Pro:caffe-master Tinaht1$
```

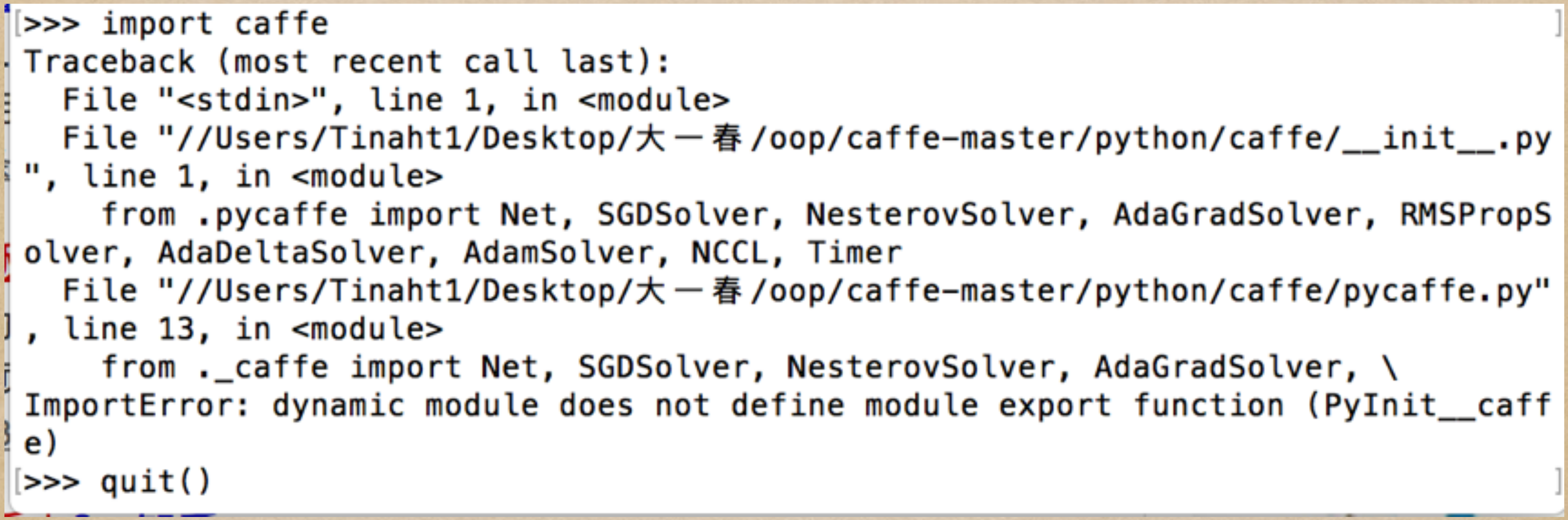
- maybe no way out

Compilation

- to compile python or matlab
- make pycaffe
- make matcaffe
- set paths in Makefile.config before make

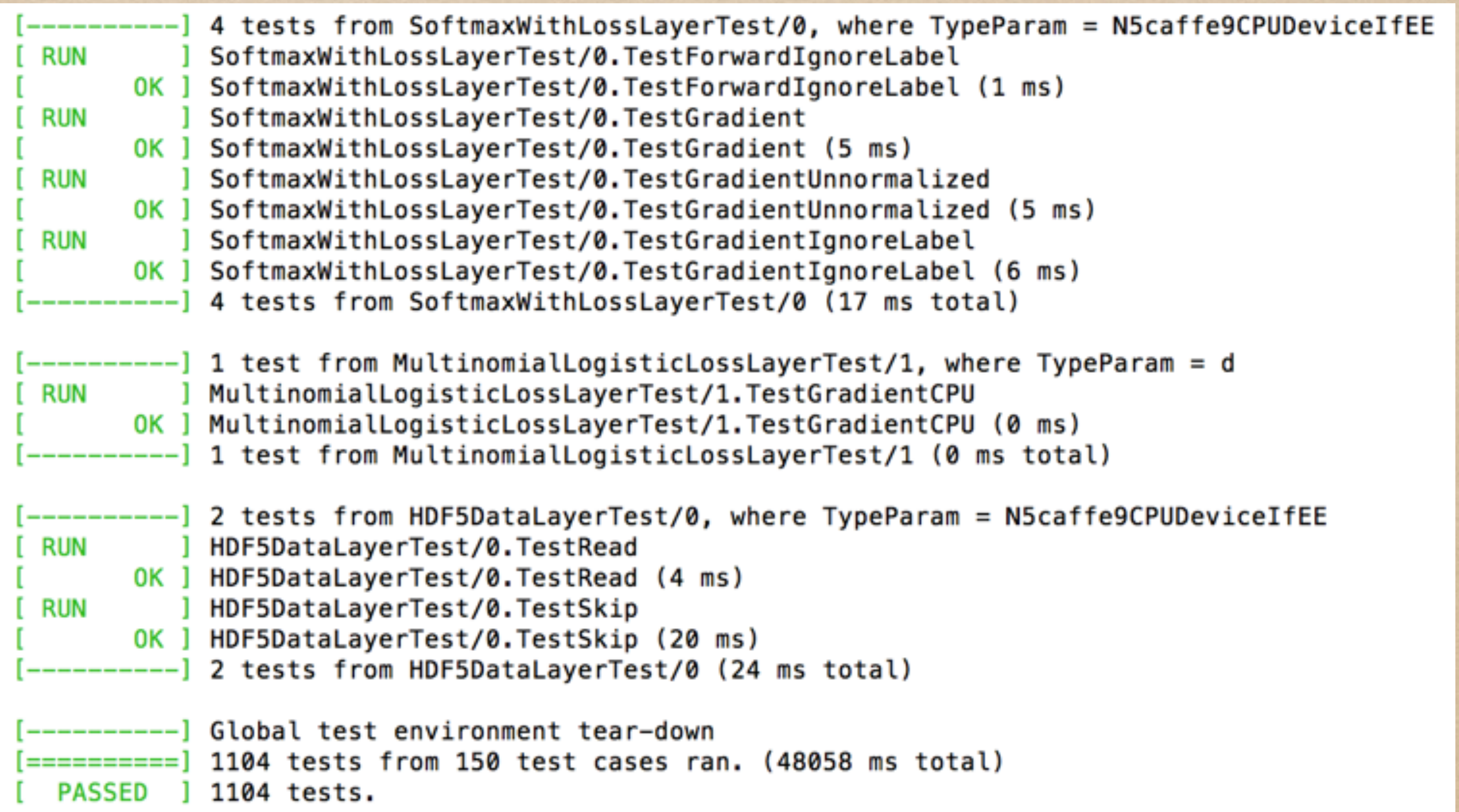
Compilation

- python 2.7 is ok, python 3.6 isn't?



```
>>> import caffe
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "//Users/Tinaht1/Desktop/大一春/oop/caffe-master/python/caffe/__init__.py
", line 1, in <module>
    from .pycaffe import Net, SGDSolver, NesterovSolver, AdaGradSolver, RMSPropS
olver, AdaDeltaSolver, AdamSolver, NCCL, Timer
  File "//Users/Tinaht1/Desktop/大一春/oop/caffe-master/python/caffe/pycaffe.py"
, line 13, in <module>
    from ._caffe import Net, SGDSolver, NesterovSolver, AdaGradSolver, \
ImportError: dynamic module does not define module export function (PyInit__caff
e)
>>> quit()
```

Compilation



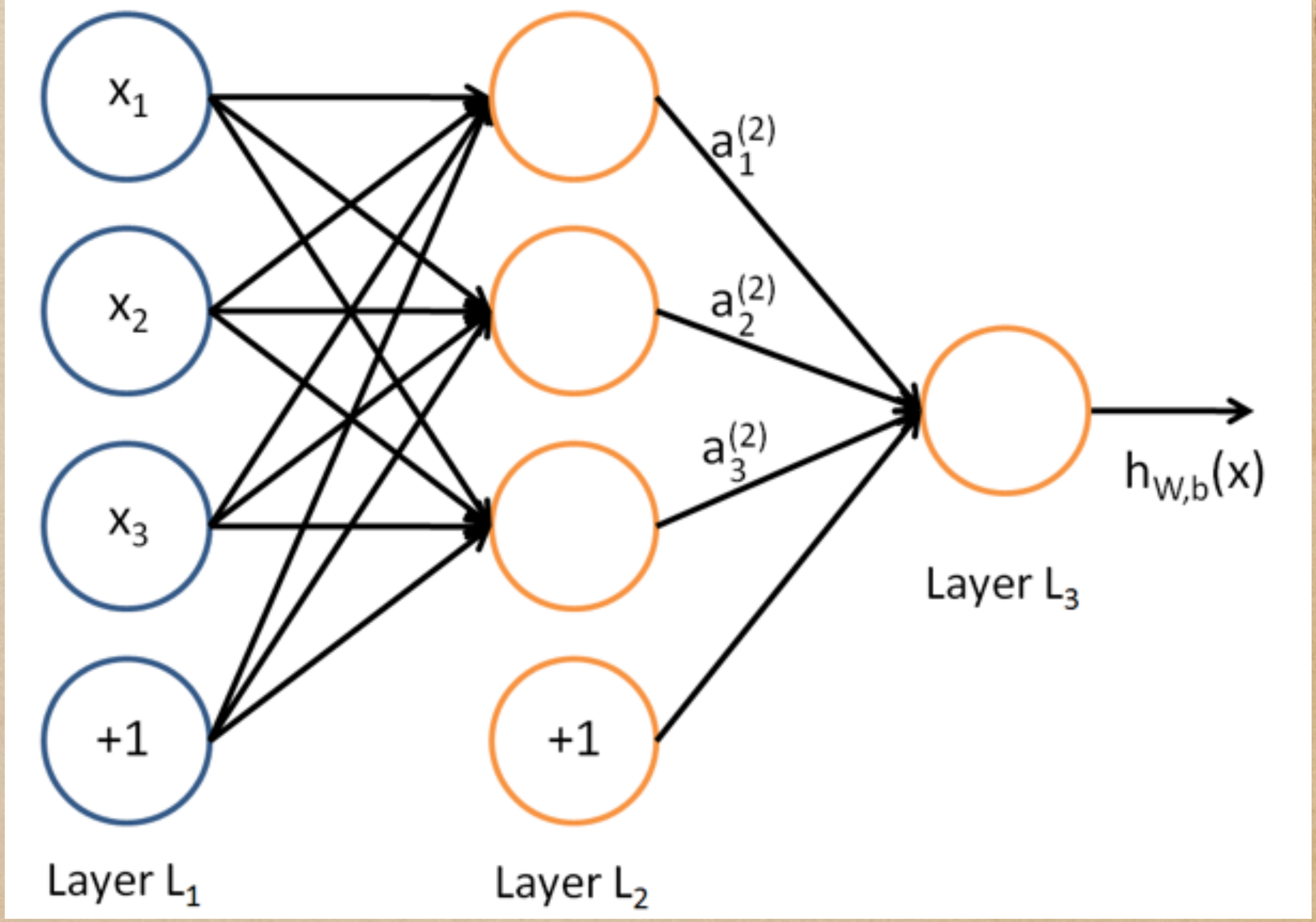
```
[----------] 4 tests from SoftmaxWithLossLayerTest/0, where TypeParam = N5caffe9CPUDeviceIfEE
[ RUN      ] SoftmaxWithLossLayerTest/0.TestForwardIgnoreLabel
[       OK ] SoftmaxWithLossLayerTest/0.TestForwardIgnoreLabel (1 ms)
[ RUN      ] SoftmaxWithLossLayerTest/0.TestGradient
[       OK ] SoftmaxWithLossLayerTest/0.TestGradient (5 ms)
[ RUN      ] SoftmaxWithLossLayerTest/0.TestGradientUnnormalized
[       OK ] SoftmaxWithLossLayerTest/0.TestGradientUnnormalized (5 ms)
[ RUN      ] SoftmaxWithLossLayerTest/0.TestGradientIgnoreLabel
[       OK ] SoftmaxWithLossLayerTest/0.TestGradientIgnoreLabel (6 ms)
[----------] 4 tests from SoftmaxWithLossLayerTest/0 (17 ms total)

[----------] 1 test from MultinomialLogisticLossLayerTest/1, where TypeParam = d
[ RUN      ] MultinomialLogisticLossLayerTest/1.TestGradientCPU
[       OK ] MultinomialLogisticLossLayerTest/1.TestGradientCPU (0 ms)
[----------] 1 test from MultinomialLogisticLossLayerTest/1 (0 ms total)

[----------] 2 tests from HDF5DataLayerTest/0, where TypeParam = N5caffe9CPUDeviceIfEE
[ RUN      ] HDF5DataLayerTest/0.TestRead
[       OK ] HDF5DataLayerTest/0.TestRead (4 ms)
[ RUN      ] HDF5DataLayerTest/0.TestSkip
[       OK ] HDF5DataLayerTest/0.TestSkip (20 ms)
[----------] 2 tests from HDF5DataLayerTest/0 (24 ms total)

[----------] Global test environment tear-down
[==========] 1104 tests from 150 test cases ran. (48058 ms total)
[  PASSED  ] 1104 tests.
```

cmake

- if you fail to make it, use cmake
- if it doesn't work either,
- keep calm, uninstall them and go back to the first page.

Deep Learning

- NN training
  each layer's
  parameter
  can be replaced by
  its
  last layer's
  "σ(w*x+b)"



Deep Learning

the first layer's w*x+b → the last layer's h(x)

$$h(x)=\sigma(\Sigma wixi+b)=\sigma(w\ x)$$

Training LeNet on MNIST

Training LeNet on MNIST

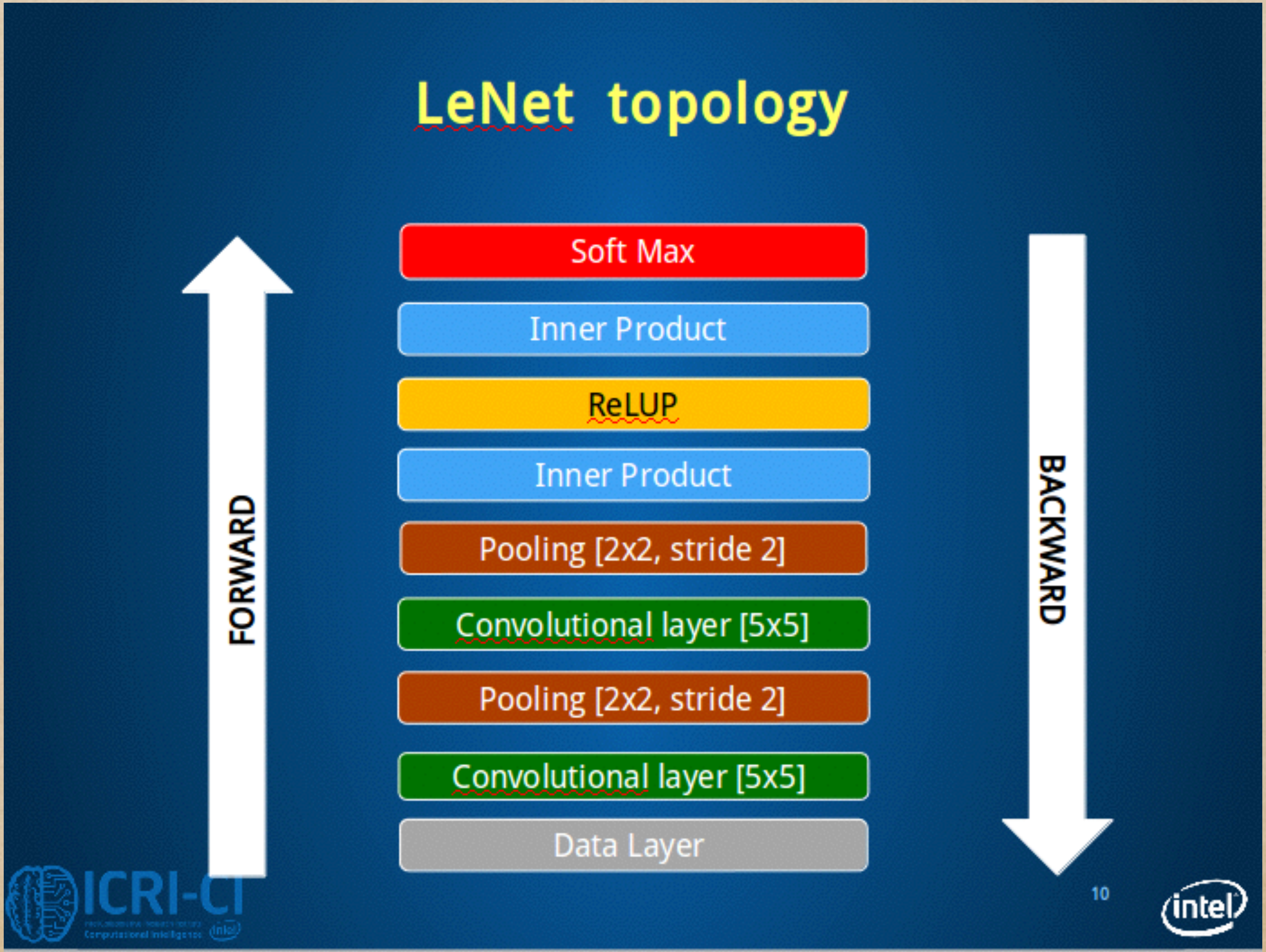
Prepare Datasets

- under “caffe_root”
- ./data/mnist/get_mnist.sh
- ./example/mnist/create_mnist.sh
- maybe wget isn’t installed: brew install wget

Define the MNIST Network

- define by protobuf :
- src/caffe/proto/caffe.proto
- Seven types of Layers(defined by prototxt)
- /examples/mnist/lenet_train_test.prototxt
- Each layer consists of two parts:
- type & parameter

Layers



LeNet topology

Soft Max

Inner Product

ReLUP

Inner Product

Pooling [2x2, stride 2]

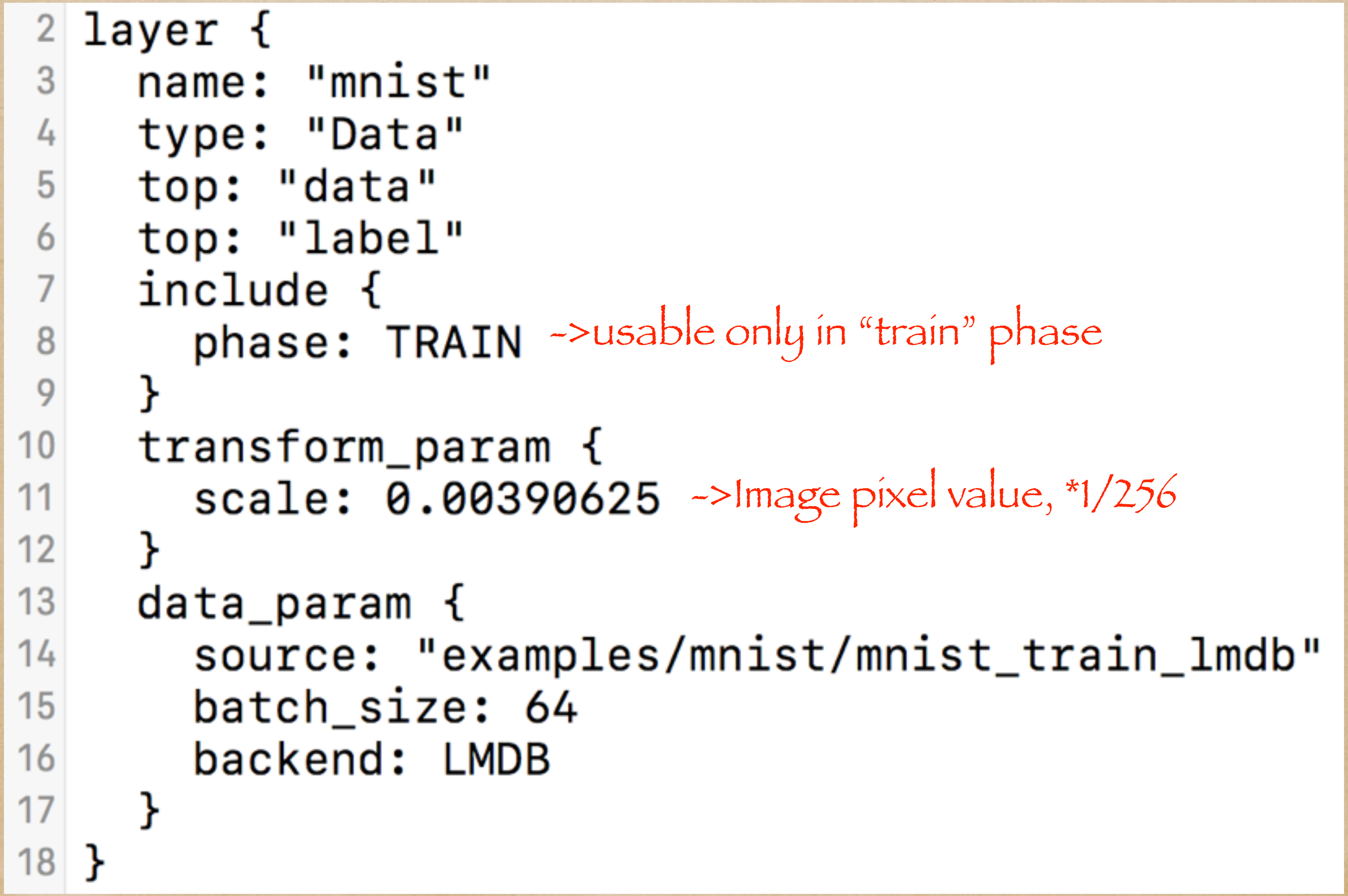
Convolutional layer [5x5]

Pooling [2x2, stride 2]

Convolutional layer [5x5]

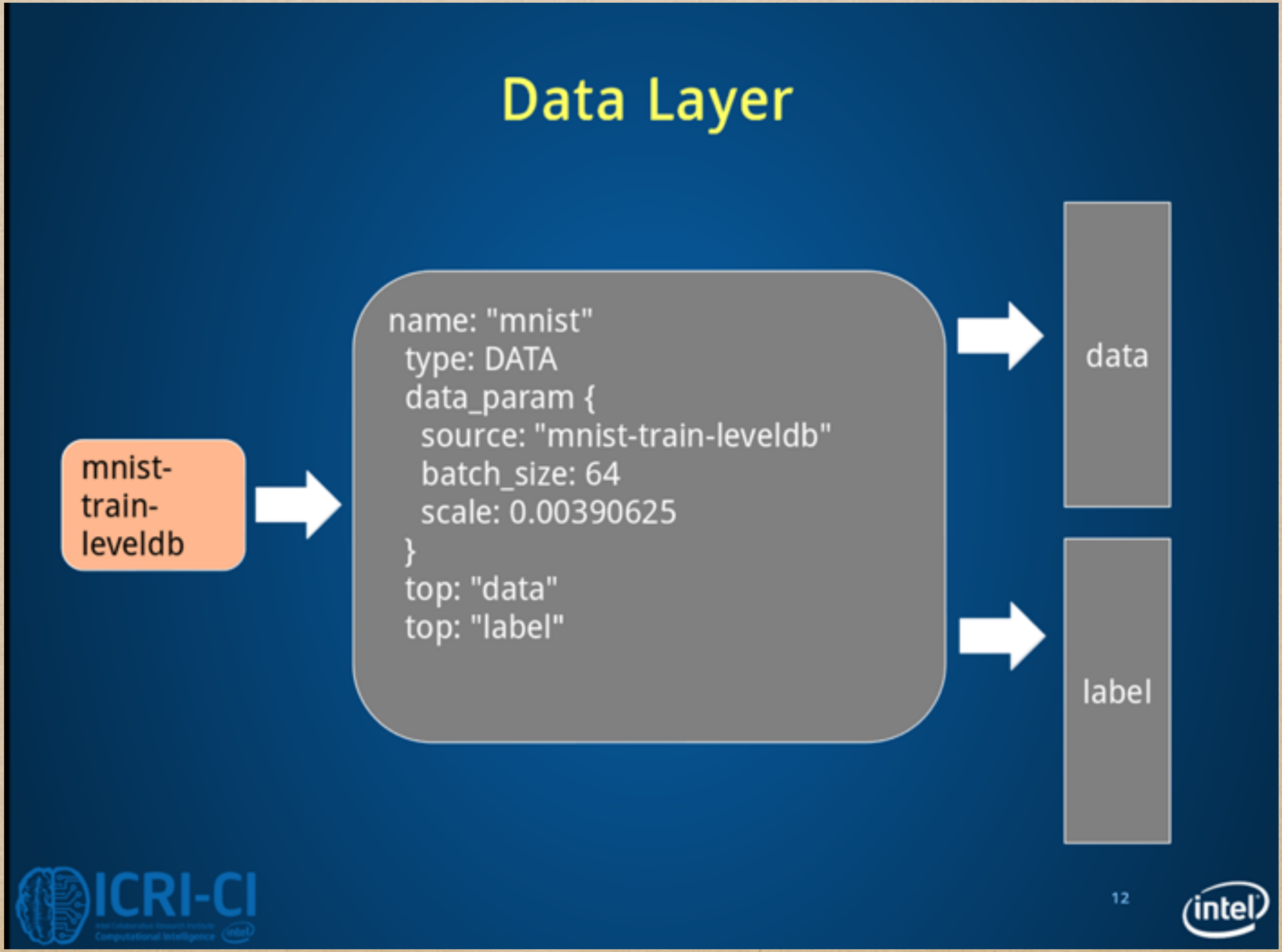
Data Layer

The Data Layer



```
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN ->usable only in "train" phase
  }
  transform_param {
    scale: 0.00390625 ->Image pixel value, *1/256
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
```

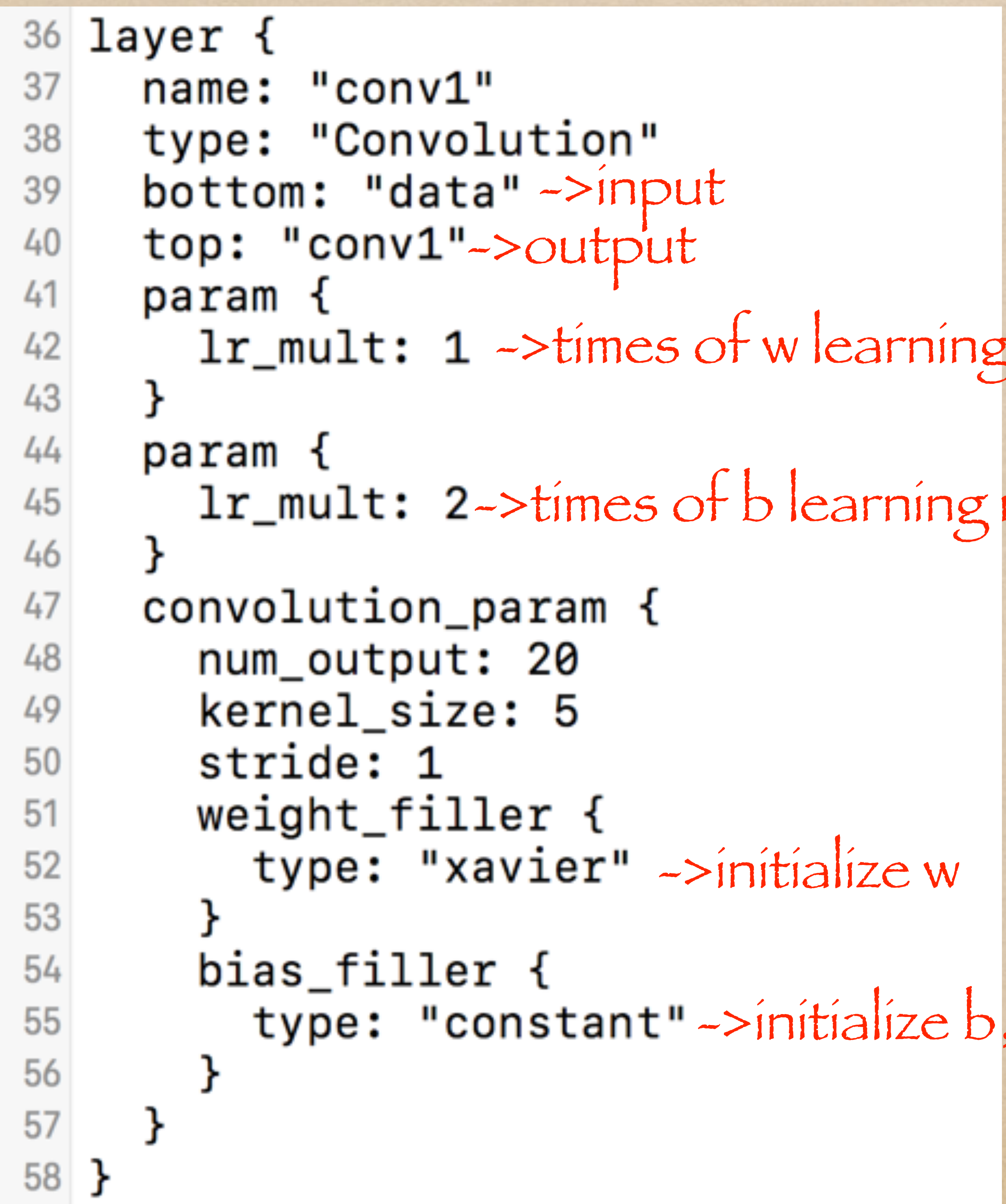
The Data Layer



Data Layer

```
name: "mnist"
  type: DATA
  data_param {
    source: "mnist-train-leveldb"
    batch_size: 64
    scale: 0.00390625
  }
  top: "data"
  top: "label"
```

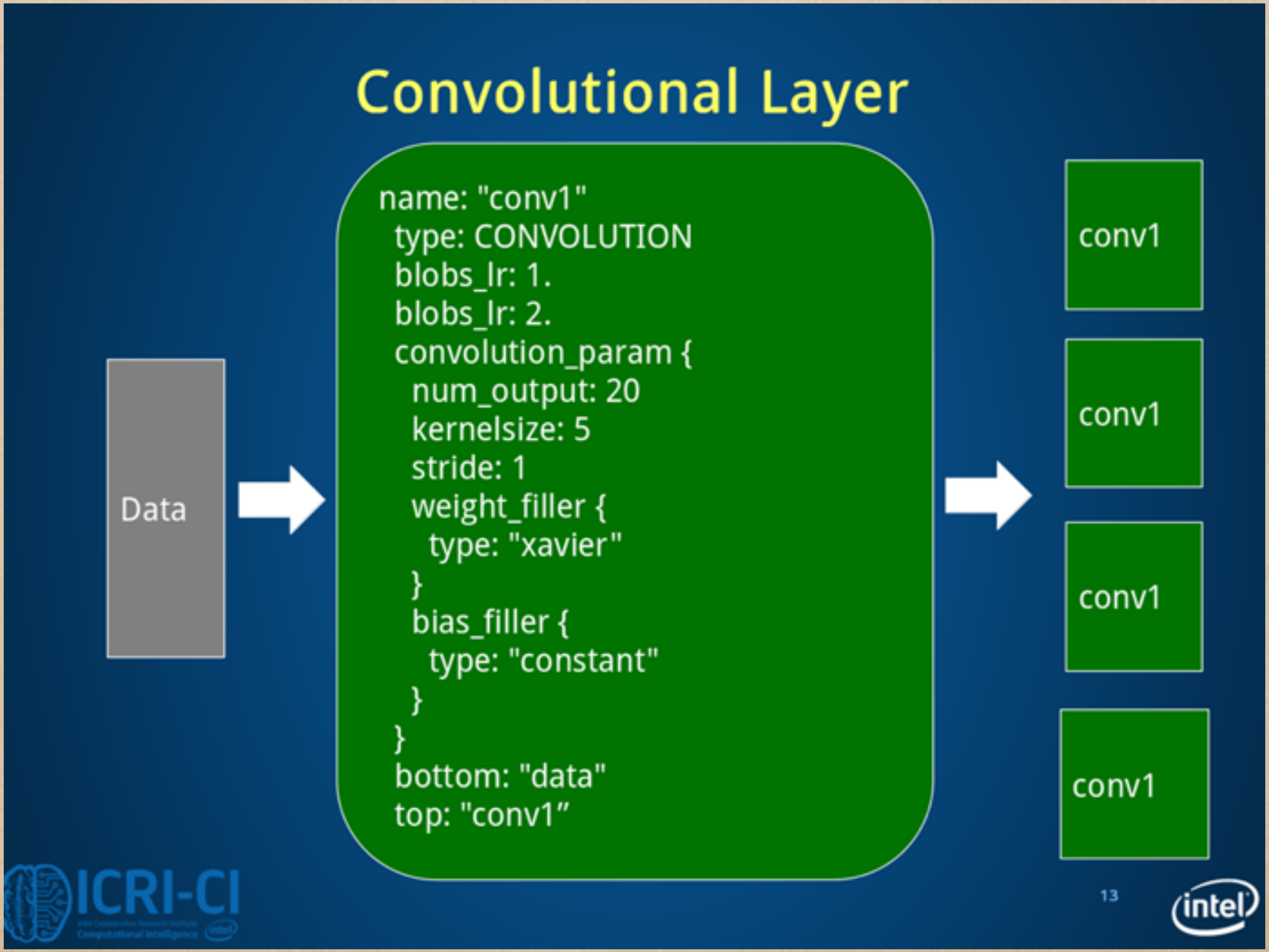
The Convolution Layer



```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data" ->input
  top: "conv1"->output
  param {
    lr_mult: 1 ->times of w learning rate
  }
  param {
    lr_mult: 2->times of b learning rate
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"  ->initialize w
    }
    bias_filler {
      type: "constant"->initialize b,   const
    }
  }
}
```

$a_1 = \sigma(b + w * a_0)$

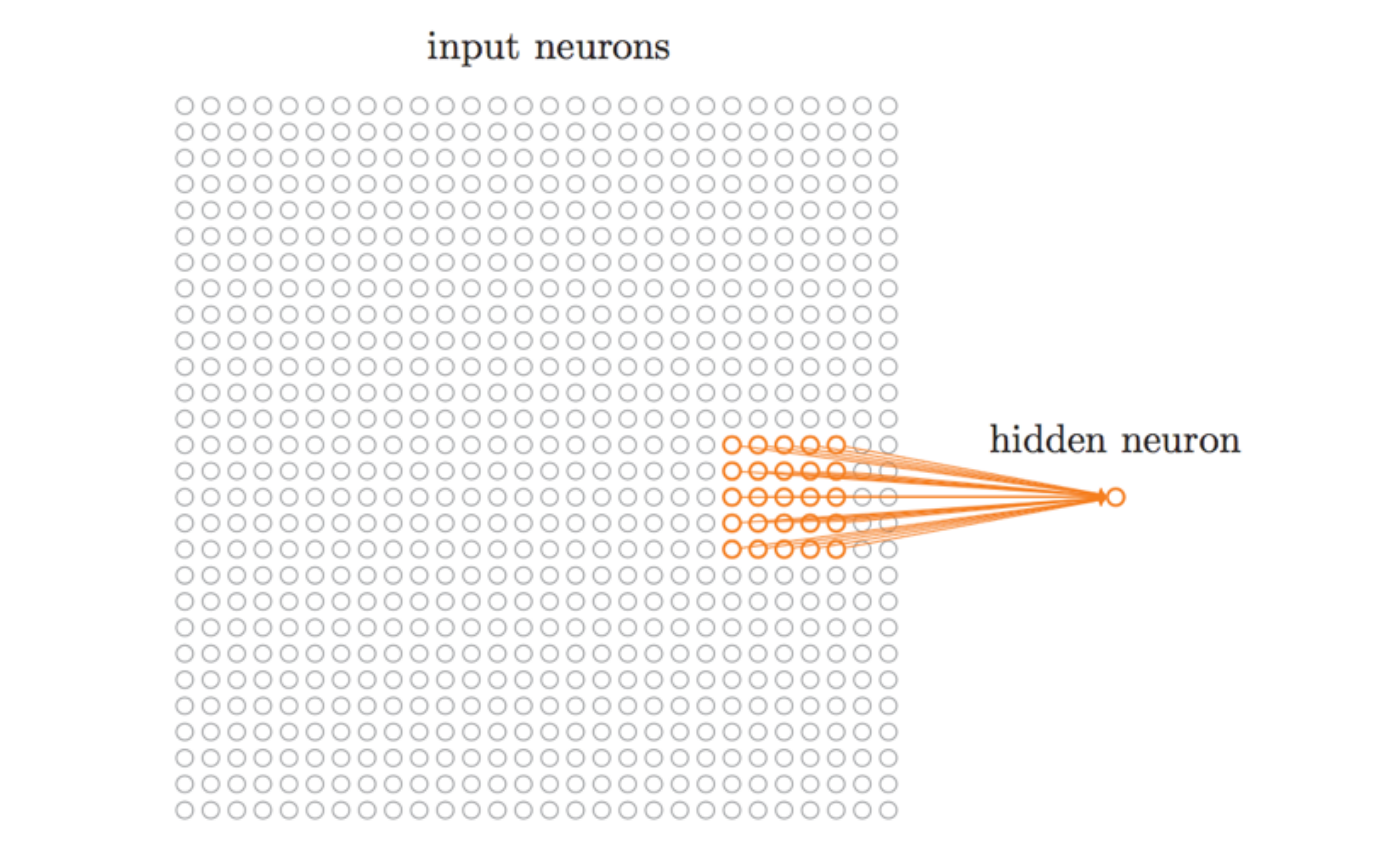
The Convolution Layer



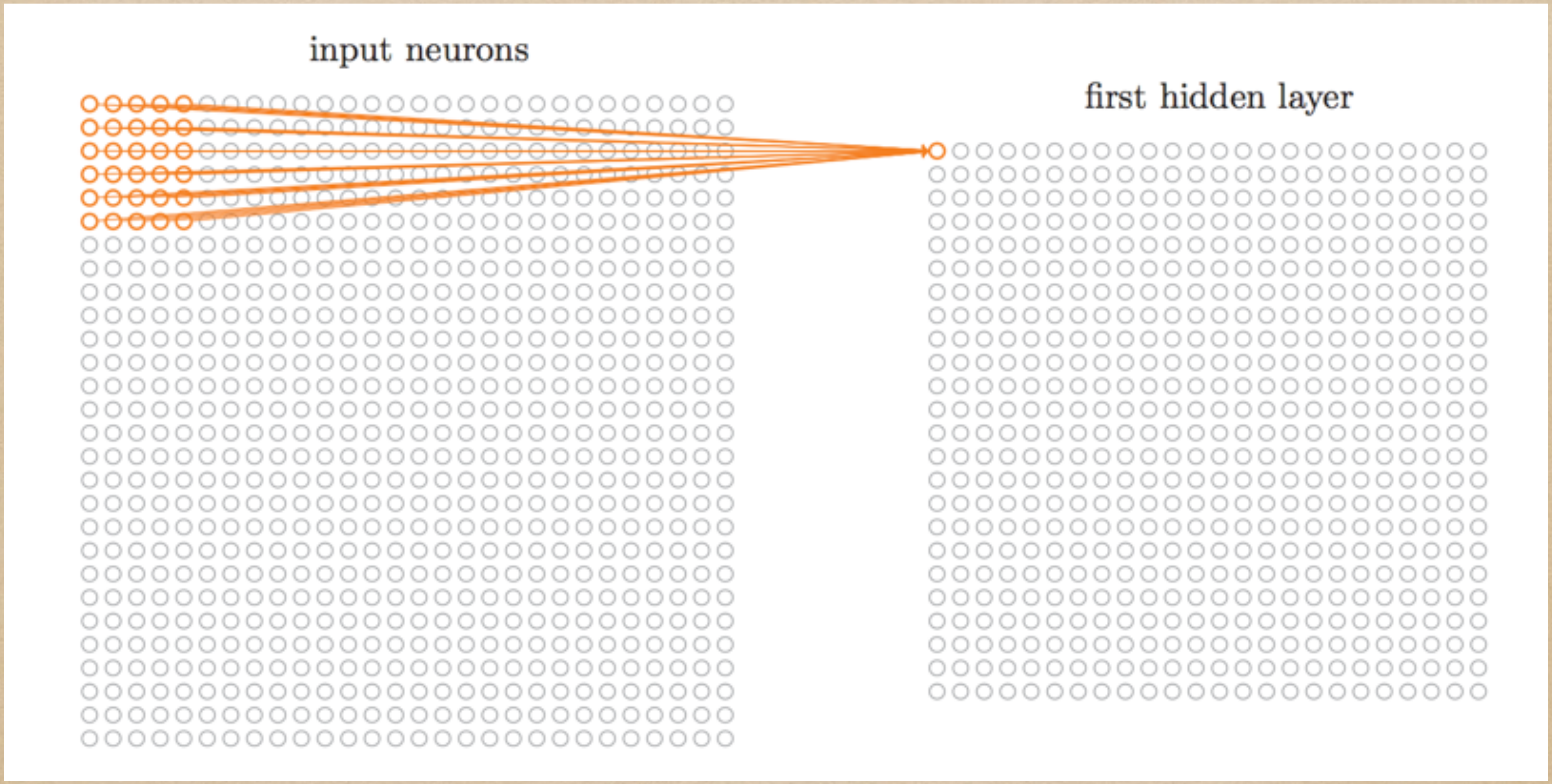
Convolutional Layer

```
name: "conv1"
  type: CONVOLUTION
  blobs_lr: 1.
  blobs_lr: 2.
  convolution_param {
    num_output: 20
    kernelsize: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
```

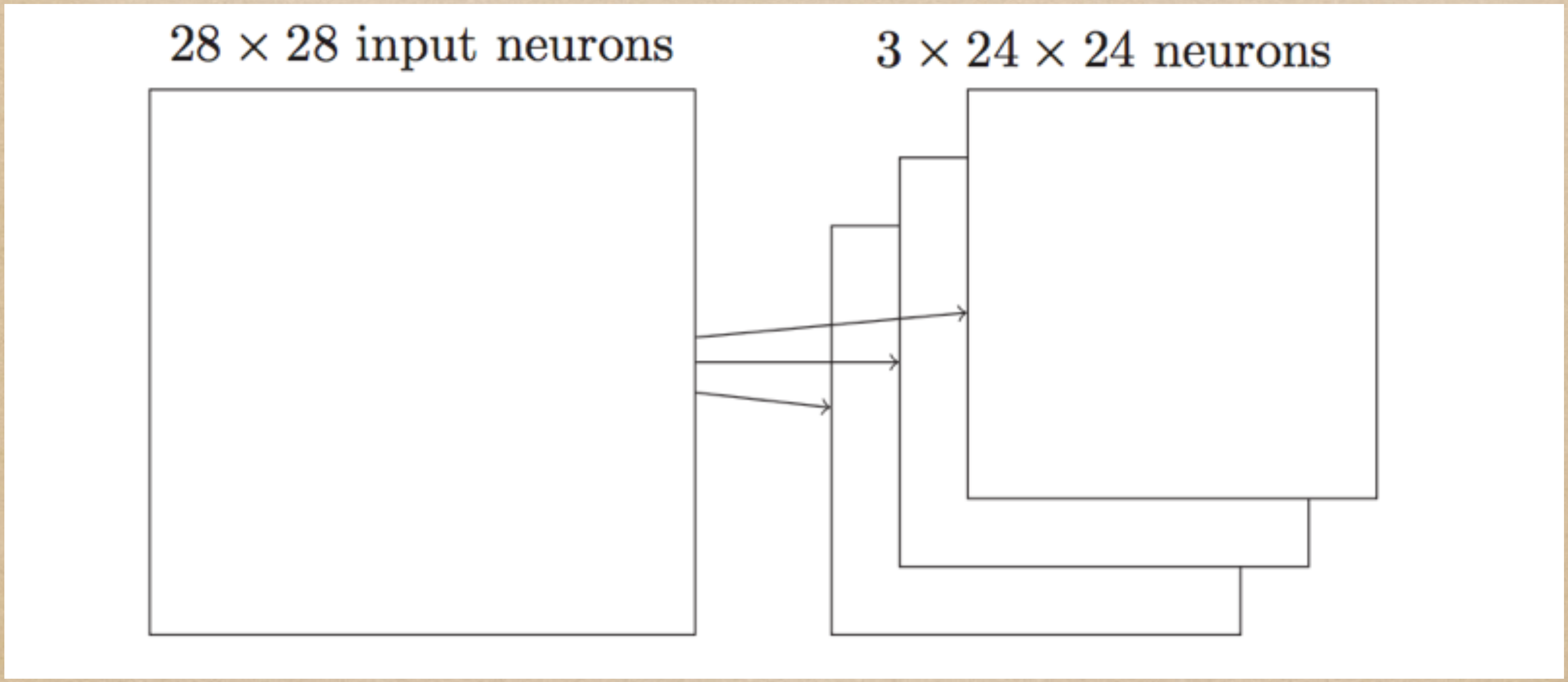
The Convolution Layer



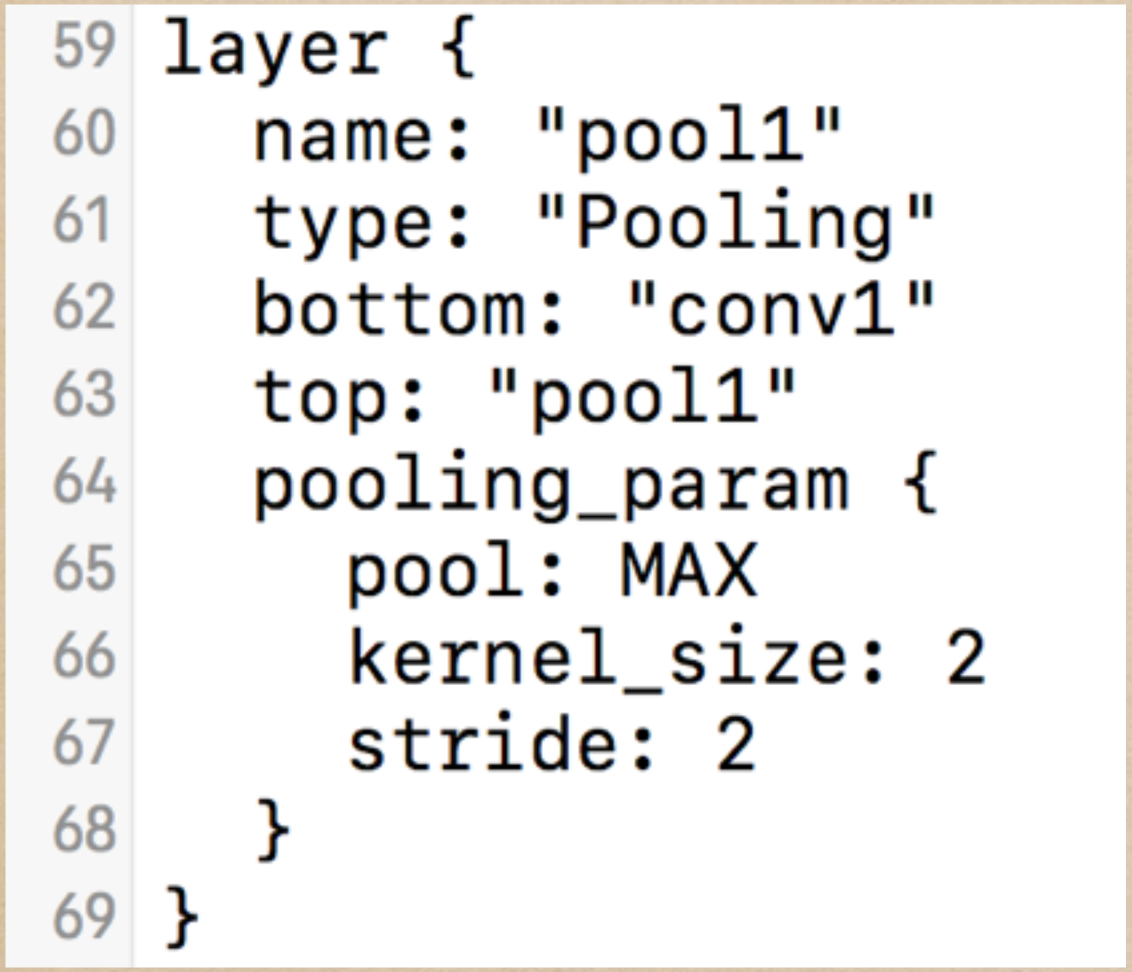
The Convolution Layer



The Convolution Layer



the Pooling Layer



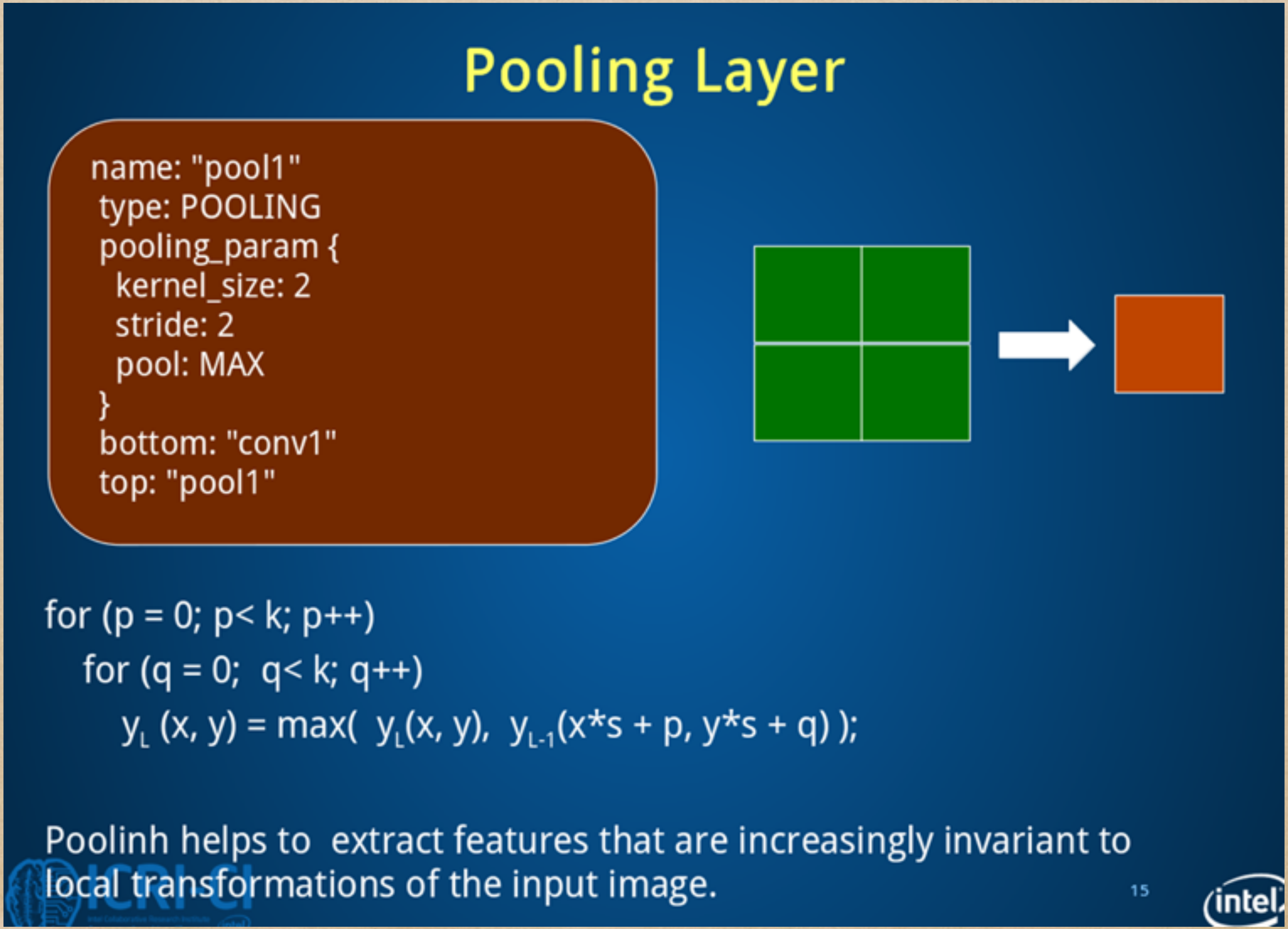
```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

L2 pooling

max pooling

->no overlapping

the Pooling Layer



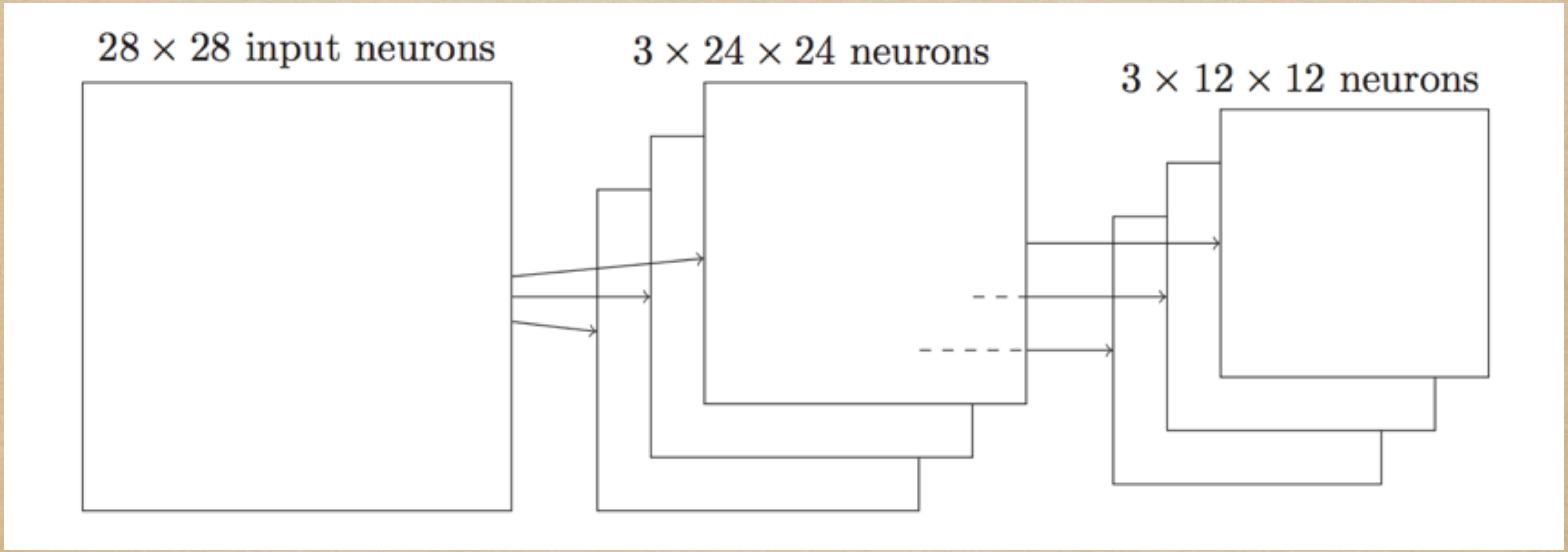
Pooling Layer

```
name: "pool1"
 type: POOLING
 pooling_param {
  kernel_size: 2
  stride: 2
  pool: MAX
 }
 bottom: "conv1"
 top: "pool1"
```

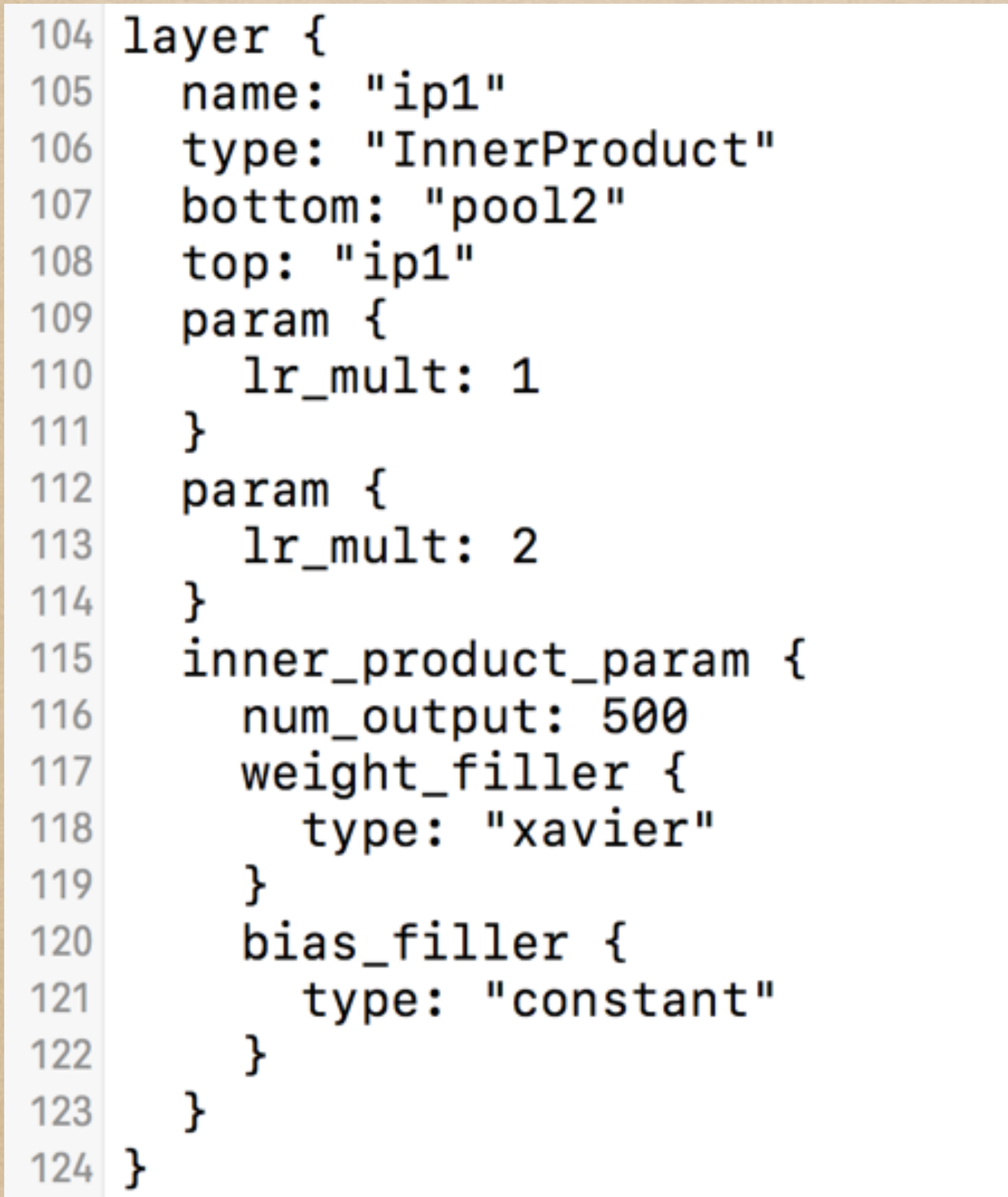
```
for (p = 0; p< k; p++)
   for (q = 0;  q< k; q++)
      $y_L (x, y) = \max( y_L(x, y), y_{L-1}(x*s + p, y*s + q) );$
```

Poolinh helps to extract features that are increasingly invariant to local transformations of the input image.

the Pooling Layer



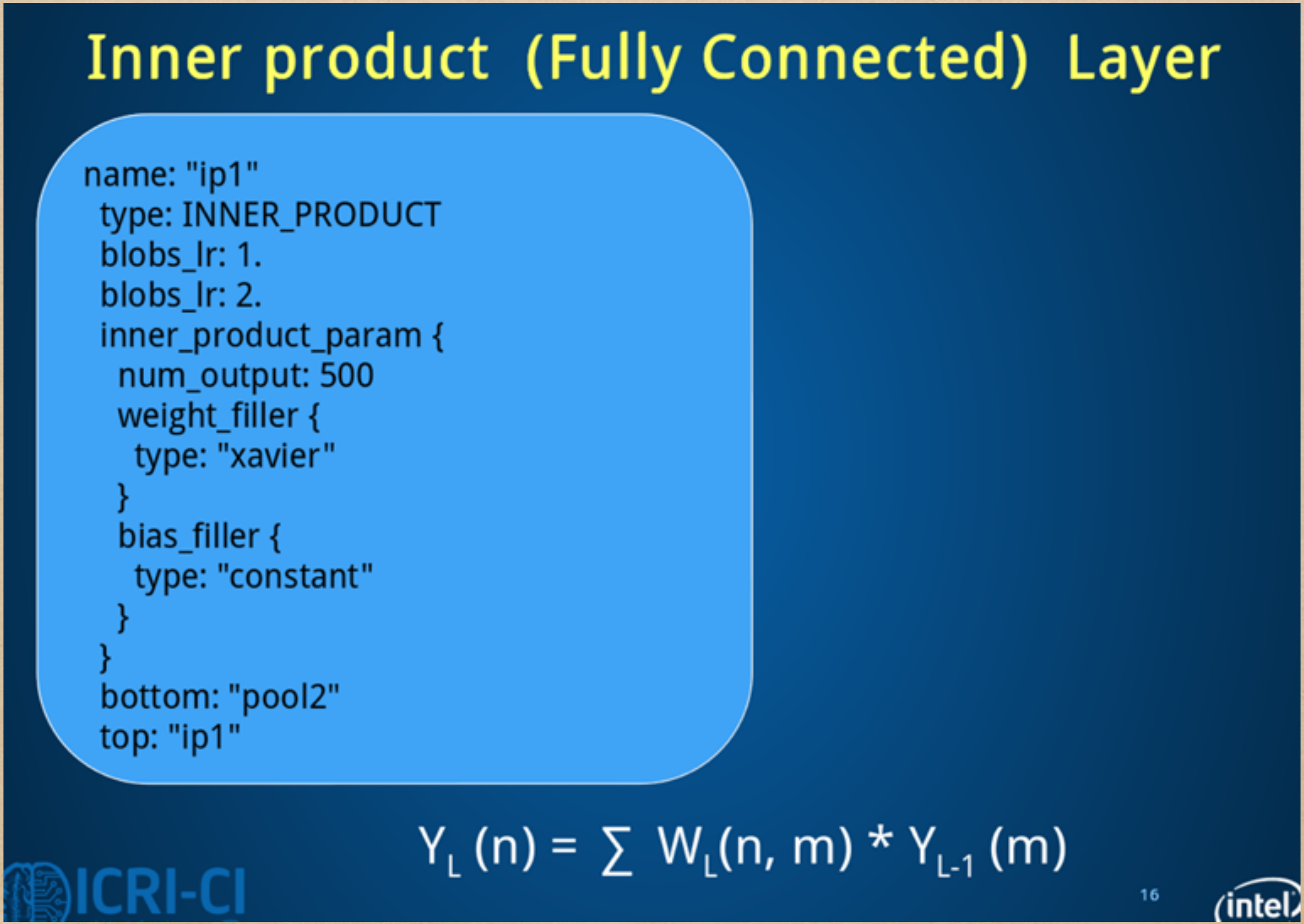
the Fully Connected Layer



```
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

change multiple dimensions into N*1*1

the Fully Connected Layer

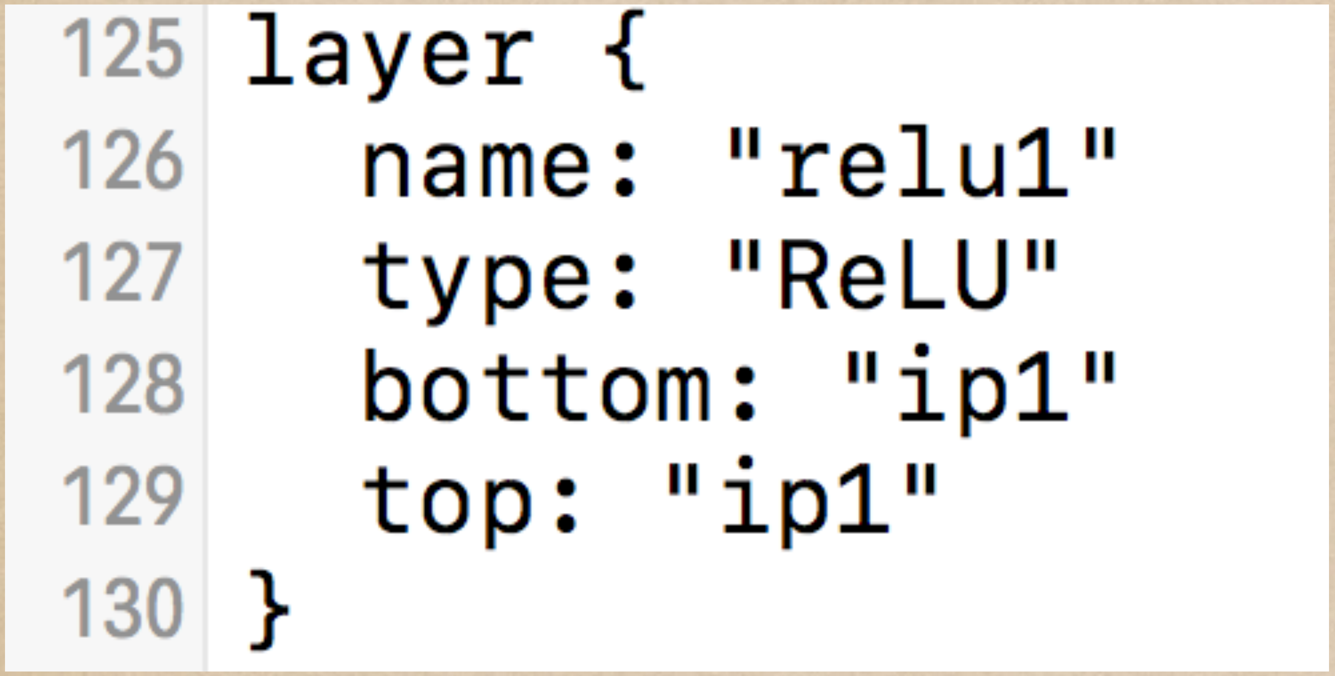


Inner product (Fully Connected) Layer

```
name: "ip1"
 type: INNER_PRODUCT
 blobs_lr: 1.
 blobs_lr: 2.
 inner_product_param {
  num_output: 500
  weight_filler {
   type: "xavier"
  }
  bias_filler {
   type: "constant"
  }
 }
 bottom: "pool2"
 top: "ip1"
```

$$Y_L(n) = \sum W_L(n, m) * Y_{L-1}(m)$$

The ReLU Layer



```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
```

the same name to the bottom and top blobs

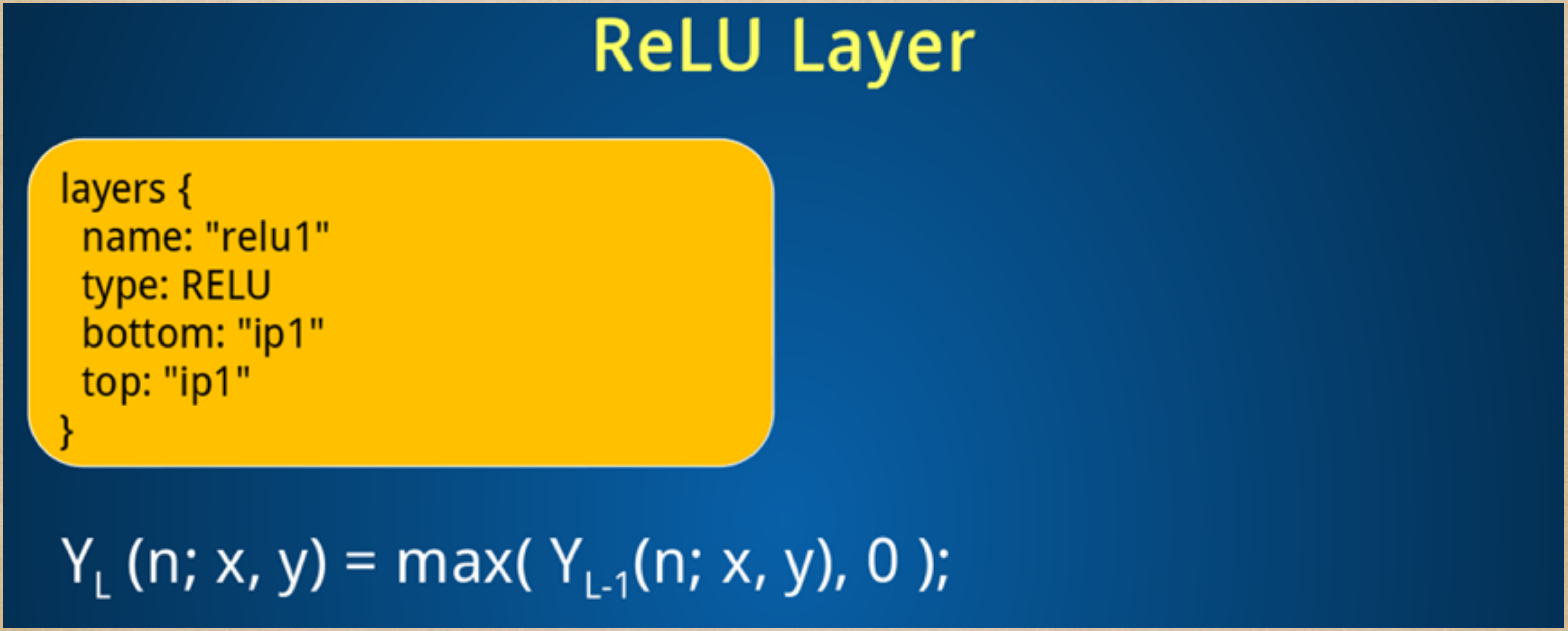
in-place operations to save some memory

After the ReLU layer, another innerproduct layer

Add:

negative_slope

The ReLU Layer



ReLU Layer

```
layers {
  name: "relu1"
  type: RELU
  bottom: "ip1"
  top: "ip1"
}
```

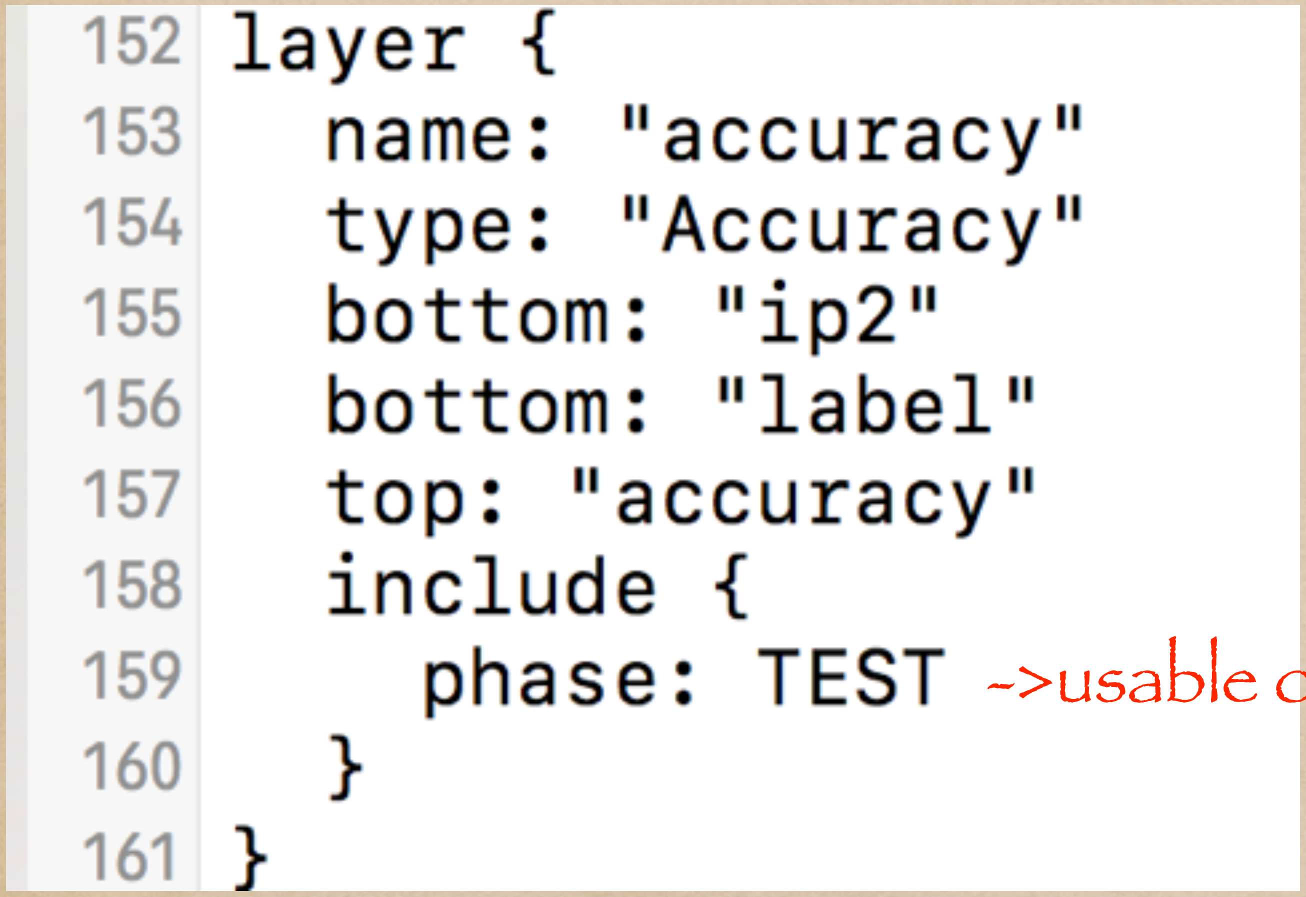
$Y_L\ (n; x, y) = \max(\ Y_{L-1}(n; x, y), 0\ );$

$h(x)=\sigma(\Sigma wixi+b)=\sigma(w\ x)$

PReLU?

```
type:”PReLU”
…
prelu_param{
  filler: {
      value: 0.25 (default)
  }
  channel_shared: false
}
```

the Accuracy Layer

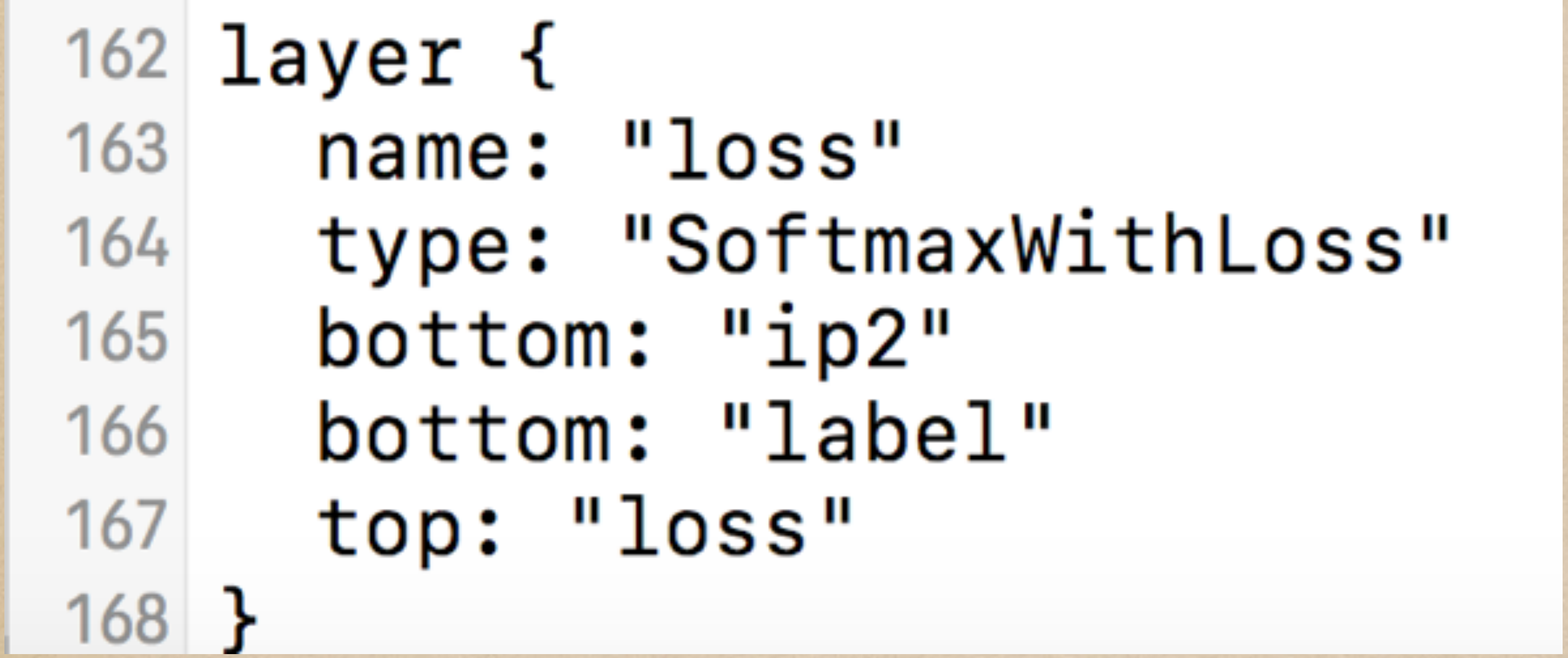


```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
```

->usable only in "test" phase

report the model accuracy

the Softmax Layer



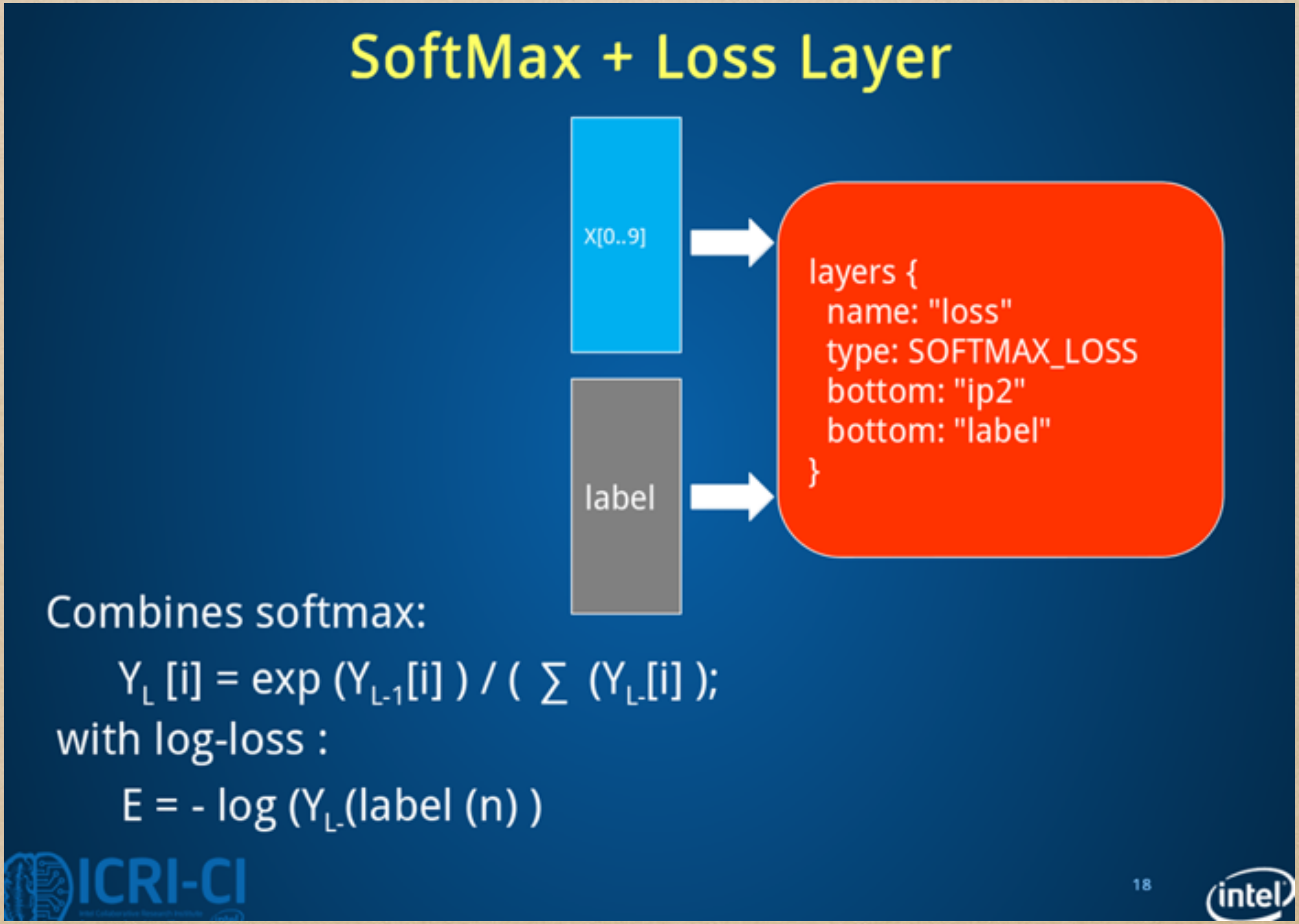
```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

to compute the loss function value

two blobs: ip2(prediction)

& label(provided by data)

the Loss Layer



SoftMax + Loss Layer

X[0..9]

label

```
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
}
```

Combines softmax:

$Y_L [i] = exp (Y_{L-1}[i]) / (\sum (Y_{L-}[i]);$

with log-loss :

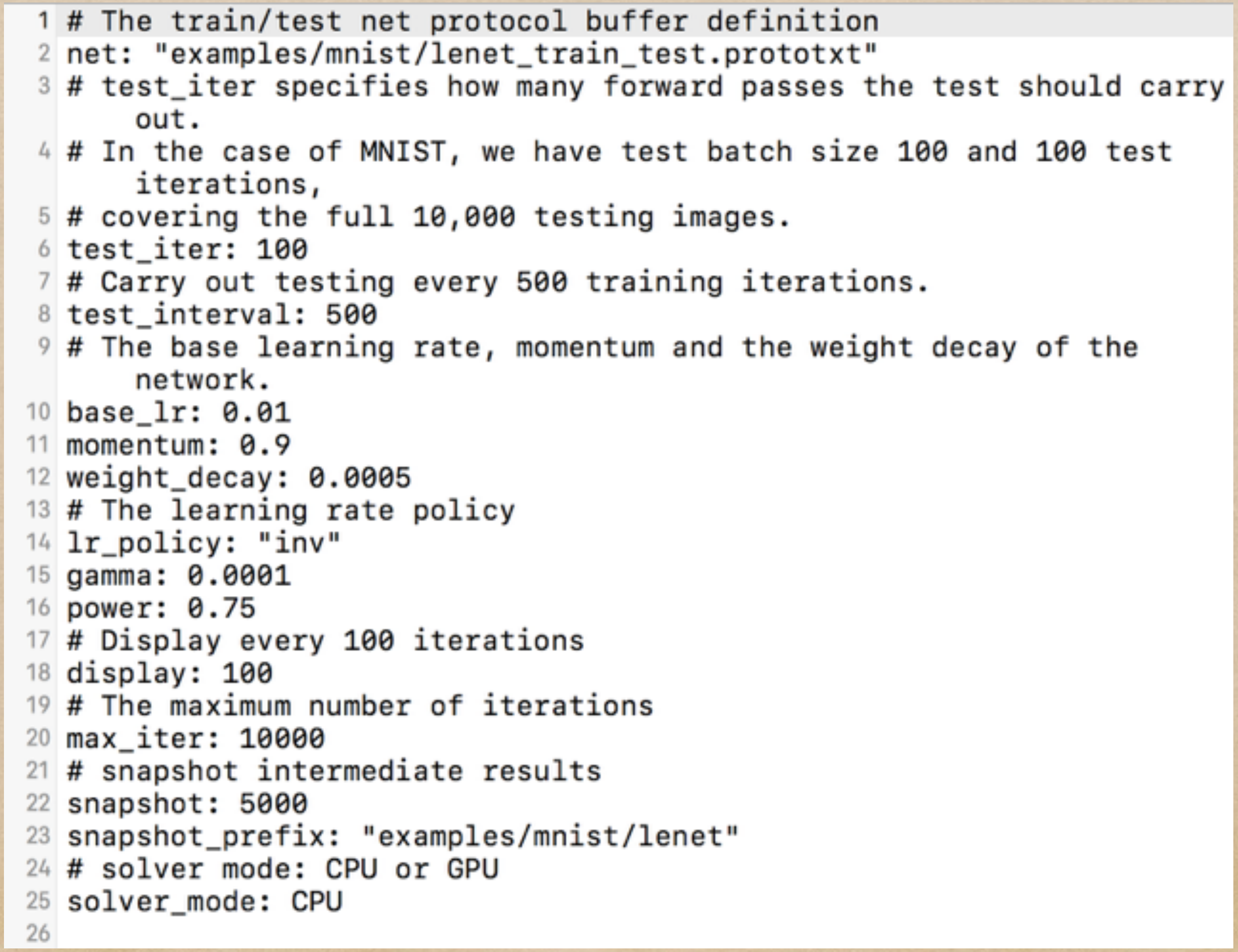
$E = - log (Y_{L-}(label (n))$

ICRI-CI

intel

Define the MNIST Solver

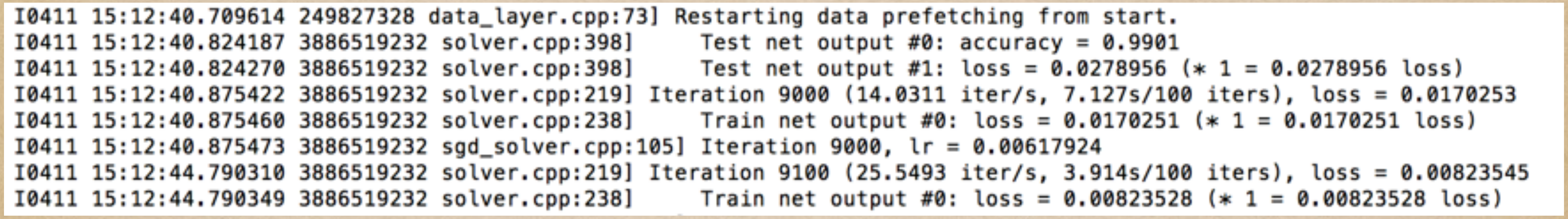
/examples/mnist/lenet_solver.prototxt



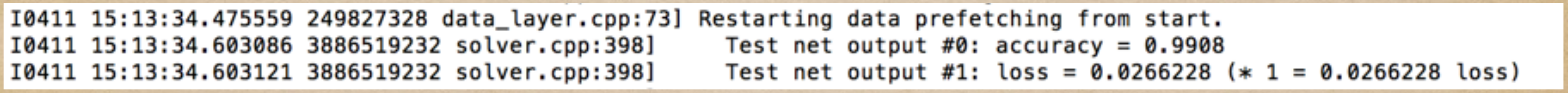
```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 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: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: CPU

```

Training and Testing the Model

./examples/mnist/train_lenet.sh



```
I0411 15:12:40.709614 249827328 data_layer.cpp:73] Restarting data prefetching from start.
I0411 15:12:40.824187 3886519232 solver.cpp:398]     Test net output #0: accuracy = 0.9901
I0411 15:12:40.824270 3886519232 solver.cpp:398]     Test net output #1: loss = 0.0278956 (* 1 = 0.0278956 loss)
I0411 15:12:40.875422 3886519232 solver.cpp:219] Iteration 9000 (14.0311 iter/s, 7.127s/100 iters), loss = 0.0170253
I0411 15:12:40.875460 3886519232 solver.cpp:238]     Train net output #0: loss = 0.0170251 (* 1 = 0.0170251 loss)
I0411 15:12:40.875473 3886519232 sgd_solver.cpp:105] Iteration 9000, lr = 0.00617924
I0411 15:12:44.790310 3886519232 solver.cpp:219] Iteration 9100 (25.5493 iter/s, 3.914s/100 iters), loss = 0.00823545
I0411 15:12:44.790349 3886519232 solver.cpp:238]     Train net output #0: loss = 0.00823528 (* 1 = 0.00823528 loss)
```



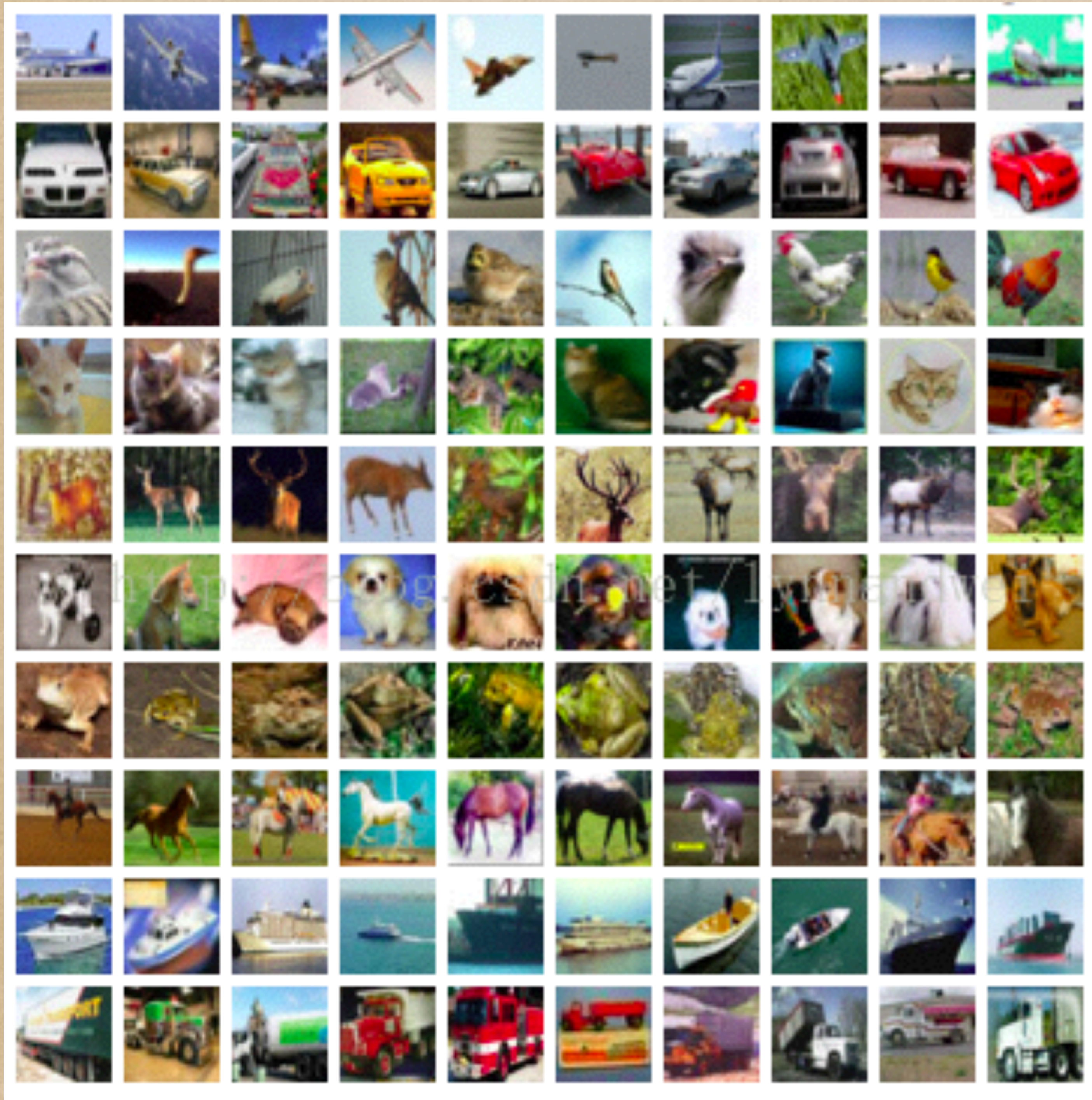
```
I0411 15:13:34.475559 249827328 data_layer.cpp:73] Restarting data prefetching from start.
I0411 15:13:34.603086 3886519232 solver.cpp:398]     Test net output #0: accuracy = 0.9908
I0411 15:13:34.603121 3886519232 solver.cpp:398]     Test net output #1: loss = 0.0266228 (* 1 = 0.0266228 loss)
```

CPU:

Time: about 10 min (10000 iterations)

Accuracy: 99%

Cifar 10



Picture Recognition

train： 50000

test： 10000

32*32

10 types

Cifar 10

- database:
- ./data/cifar10/get_cifar10.sh
- ./examples/cifar10/create_cifar10.sh
- a little problem(create_cifar10.sh should be run just under “caffe root”)

Training

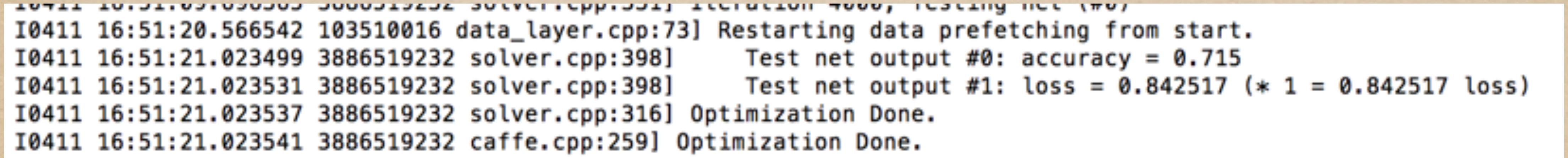
- ./examples/cifar10/train_quick.sh

- Get:

- cifar10_quick_iter_4000.caffemodel.h5 (model)

- cifar10_quick_iter_4000.solverstate.h5 (state)

Result



```
I0411 16:51:09.090309 3886519232 solver.cpp:331] Iteration 4000, Testing net (#0)
I0411 16:51:20.566542 103510016 data_layer.cpp:73] Restarting data prefetching from start.
I0411 16:51:21.023499 3886519232 solver.cpp:398]     Test net output #0: accuracy = 0.715
I0411 16:51:21.023531 3886519232 solver.cpp:398]     Test net output #1: loss = 0.842517 (* 1 = 0.842517 loss)
I0411 16:51:21.023537 3886519232 solver.cpp:316] Optimization Done.
I0411 16:51:21.023541 3886519232 caffe.cpp:259] Optimization Done.
```

- really slowly….
- CPU only:
- Time： about 20min （4000 iterations)
- Accuracy： 70%
- I Need a GPU….

using caffe training model

- 1、prepare datasets
- 2、write *_test.prototxt for modeling
- 3、write *_solver.prototxt for optimization
- 4、run it in the command line
- Then you'll get a training model.

Thank you~