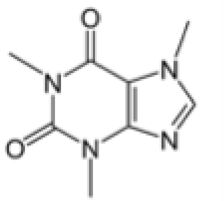
What is Caffe?

Convolutional Architecture for Fast Feature Embedding

Caffe is **a deep learning framework** made with expression, speed, and modularity in mind.
It is developed by the **Berkeley Vision and Learning Center (BVLC)** and community contributors. Yangqing Jia created the project during his PhD at UC Berkeley.



caffe.berkeleyvision.org

github.com/BVLC/caffe

http://caffe.berkeleyvision.org/tutorial/

Windows

http://caffe.berkeleyvision.org/installation.html

https://github.com/Microsoft/caffe

Linux（Ubuntu）

http://caffe.berkeleyvision.org/install_apt.html

General Dependencies

- sudo apt-get install libatlas-base-dev
- sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libboost-all-dev libhdf5-serial-dev
- sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev protobuf-compiler

Install Caffe

- git clone https://github.com/BVLC/caffe.git
- cd caffe
- cp Makefile.config.example Makefile.config

CPU-Only（In VMVare）

- vi Makefile.config
- # Adjust Makefile.config (for example, if using Anaconda Python)
- # uncomment CPU_ONLY := 1

Compile

- make all
- make pycaffe

Learn LeNet on MNIST

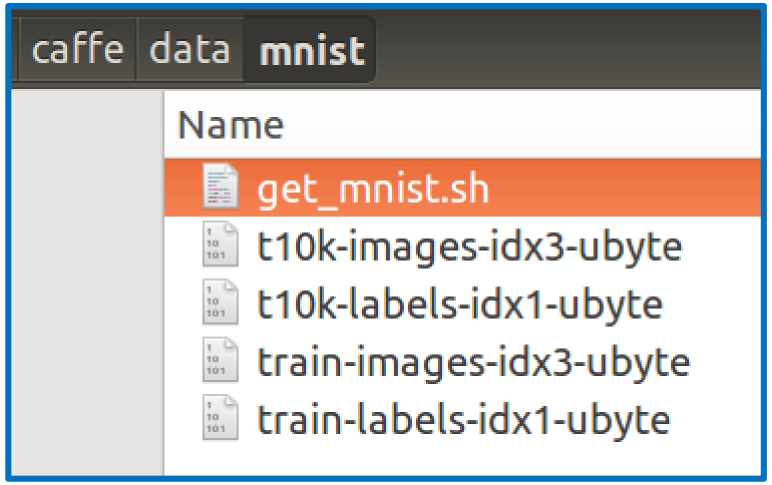
Dataset http://yann.lecun.com/exdb/mnist/

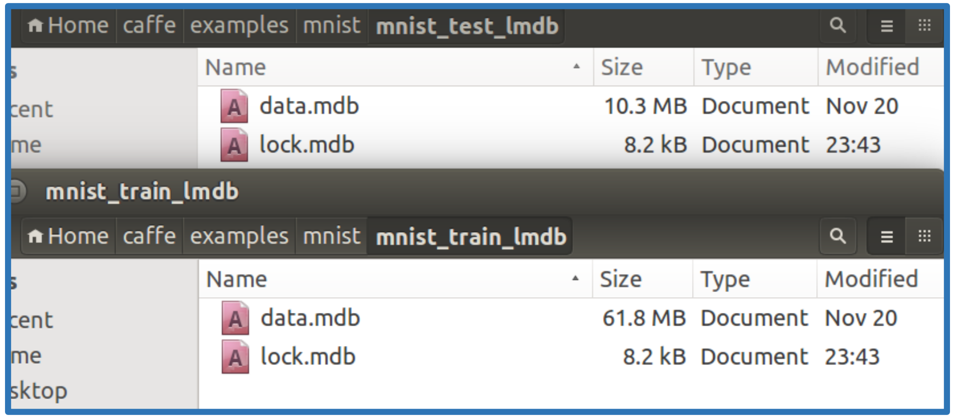
Training LeNet on MNIST with Caffe

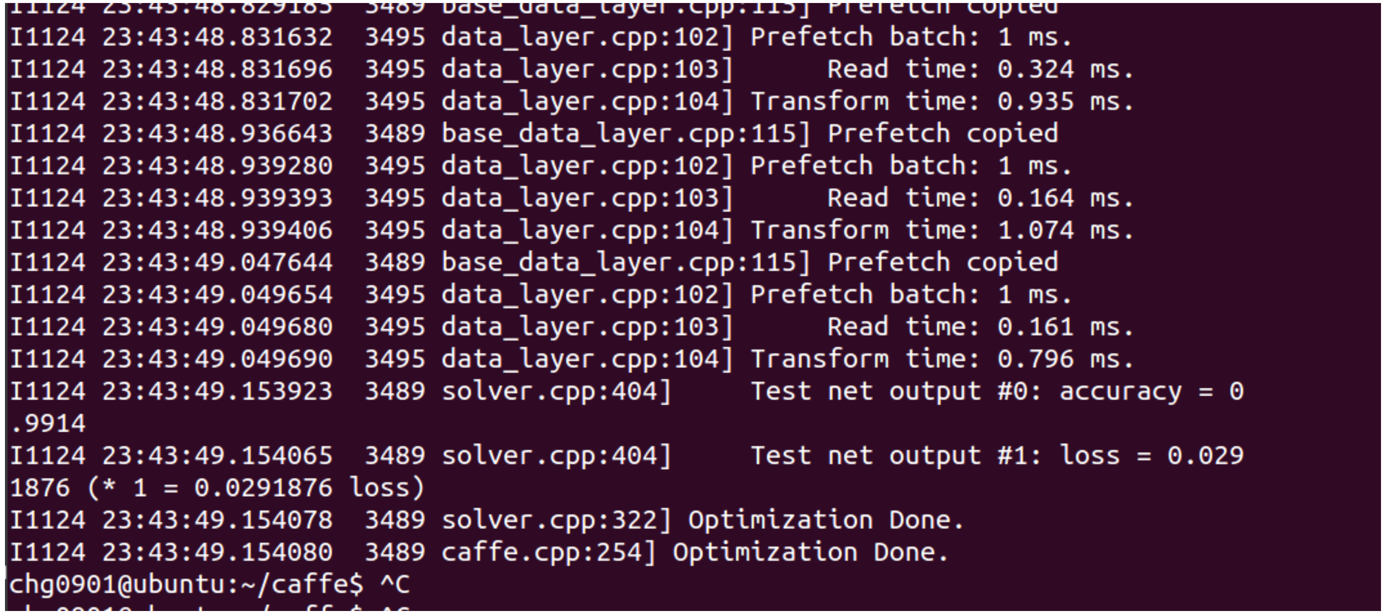
We will assume that you have Caffe successfully compiled. If not, please refer to the Installation page. In this tutorial, we will assume that your Caffe installation is located at `CAFFE_ROOT`.

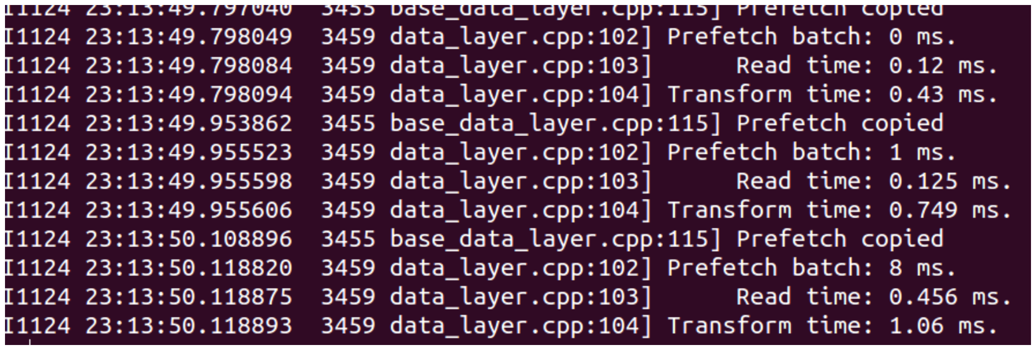
Test（Mnist)

- ❑ cd caffe
- ❑ sh data/mnist/get_mnist.sh
- ❑ sh examples/mnist/create_mnist.sh
- ❑ vi examples/mnist/lenet_solver.prototxt
- ❑ ./examples/mnist/train_lenet.sh









Why Caffe? In one sip…

- **Expression**: models + optimizations are plaintext schemas, not code.
- **Speed**: for state-of-the-art models and massive data.
- **Modularity**: to extend to new tasks and settings.
- **Openness**: common code and reference models for reproducibility.
- **Community**: joint discussion and development through BSD-2 licensing.

Solver.prototxt

https://github.com/BVLC/caffe/wiki/Solver-Prototxt

The solver.prototxt is a configuration file used to tell caffe how you want the network trained.

- # 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

```
name: "CIFAR10_quick_test"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 1 dim: 3 dim: 32 dim: 32 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 64
  }
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
  }
}
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

- Train.prototxt

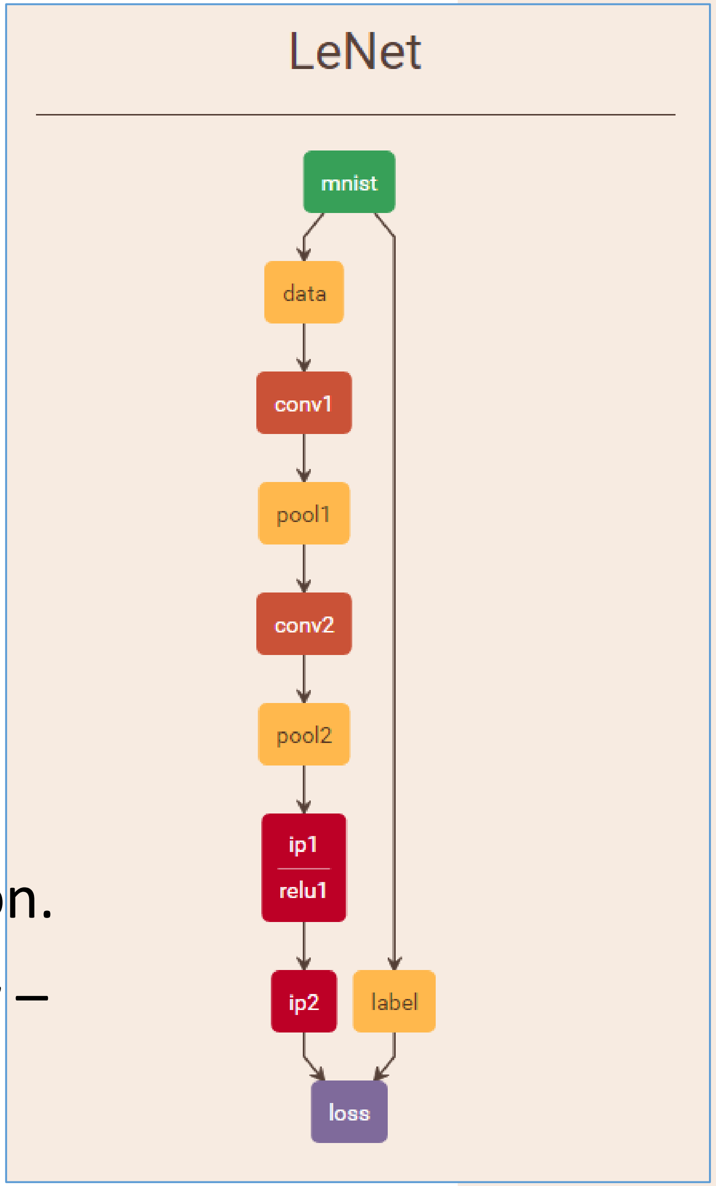
Net

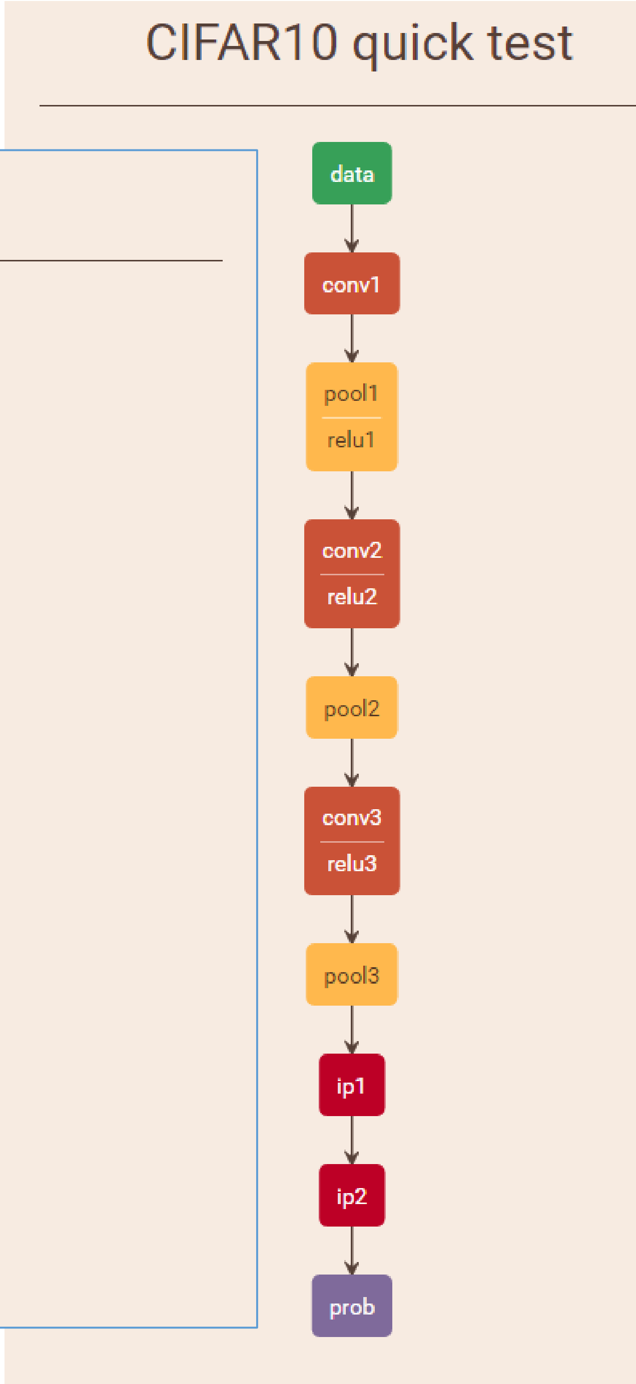
lenet_train_test.prototxt

- A network is a set of layers and their connections as a DAG:

```
name: "dummy-net"
layer { name: "data" …}
layer { name: "conv" …}
layer { name: "pool" …}
     … more layers …
layer { name: "loss" …}
```

- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as ***blobs*** – an array interface



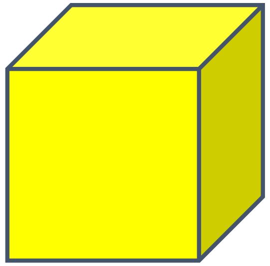


Blob

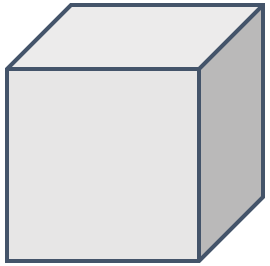
```
name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
… definition …
```

Blobs are N-D arrays for storing and communicating information.

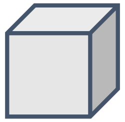
- hold data, derivatives, and parameters
- lazily allocate memory
- shuttle between CPU and GPU



Data
*N*umber x *K* Channel x *H*eight x *W*idth
256 x 3 x 227 x 227 for ImageNet train input

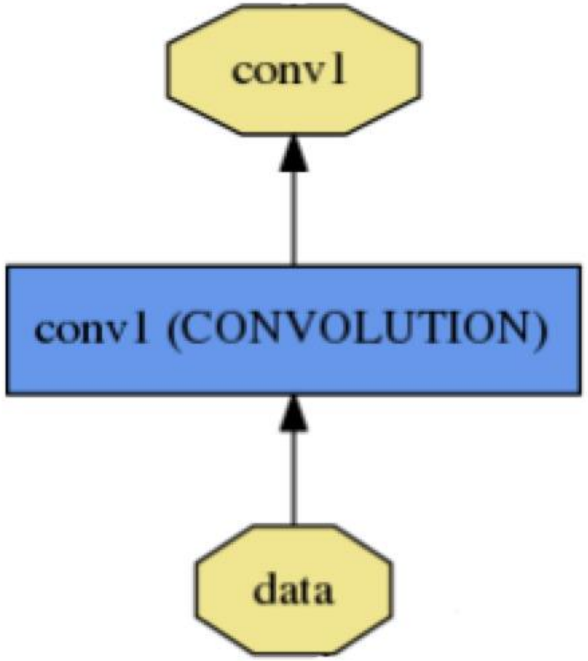


Parameter: Convolution Weight
N Output x *K* Input x *H*eight x *W*idth
96 x 3 x 11 x 11 for CaffeNet conv1



Parameter: Convolution Bias
96 x 1 x 1 x 1 for CaffeNet conv1

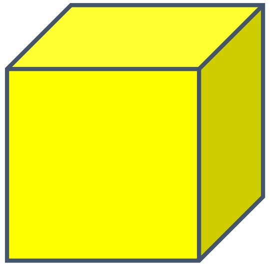
top
blob



bottom
blob

Blob

Blobs provide a unified memory interface.



Reshape(num, channel, height, width)
- declare dimensions
- make *SyncedMem* -- but only lazily allocate

cpu_data(), mutable_cpu_data()
- host memory for CPU mode
gpu_data(), mutable_gpu_data()
- device memory for GPU mode

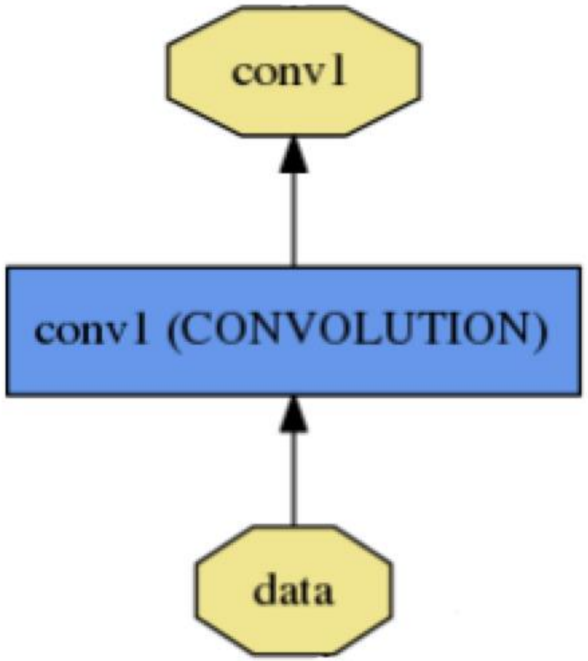
{cpu,gpu}_diff(), mutable_{cpu,gpu}_diff()
- derivative counterparts to data methods
- easy access to data + diff in forward / backward

Layer

```
name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
        type: "xavier"
    }
}
```

name, type, and the connection structure (input blobs and output blobs)

layer-specific parameters



- Every layer type defines

- **Setup**
- **Forward**
- **Backward**

- Nets + Layers are defined by protobuf schema

Data Layer

- layer {
-     name: "mnist"
-     type: "Data"
-     transform_param {
-       scale: 0.00390625   // The input pixels are normalized to [0, 1],0.00390625=1/256
-     }
-     data_param {
-       source: "mnist_train_lmdb"
-       backend: LMDB
-       batch_size: 64   //batch mode
-     }
-     top: "data“   //generate two blobs, data blob 和label blob
-     top: "label"
-   }

Convolution Layer

[An Explanation of Xavier Initialization](http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization)
http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

- layer {
- name: "conv1"
- type: "Convolution"
- //Parameter adjustment of the learning rate
- param { lr_mult: 1 } //the lR of weight is same as the solver
- param { lr_mult: 2 } //the LR of bias is twice as high as the solvers’

Convolution Layer

[An Explanation of Xavier Initialization](http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization)
http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

- convolution_param {
- num_output: 20 //number of activation map
- kernel_size: 5 //the size of filter 5*5
- stride: 1
- weight_filler {
- type: “xavier” //xavier algorithm, automatically initialize weight filler, //based on the number of input and output neurons
- }
- bias_filler {
- type: "constant“ //initialize bias filler as constant, default value 0
- }
- }
- bottom: "data“ //Data Blobs from the underlying layer are used as inputs
- top: "conv1“ // produces the `conv1` layer
- }

Pooling Layer

- layers {
- name: "pool1"
- type: POOLING
- pooling_param {
    - kernel_size: 2
    - stride: 2 //Prevent overlapping areas
    - pool: MAX
- }
- bottom: "conv1"
- top: "pool1"}

Innerprodect Layer (Fully Connection Layer)

- layer {
-   name: "ip1“ //caffe calls it an innerprodect layer
-   type: "InnerProduct"
-   bottom: "pool3"
-   top: "ip1"
-   param {
-     lr_mult: 1
-   }
-   param {
-     lr_mult: 2
-   }
-   inner_product_param {
-     num_output: 64
-   }
- }

- layer {
-   name: "ip2"
-   type: "InnerProduct"
-   bottom: "ip1"
-   top: "ip2"
-   param {
-     lr_mult: 1
-   }
-   param {
-     lr_mult: 2
-   }
-   inner_product_param {
-     num_output: 10
-   }
- }

ReLU Layer

- layer {
-   name: "relu3"
-   type: "ReLU"
-   bottom: "conv3"
-   top: "conv3"
- }

- layer {
-   name: "relu2"
-   type: "ReLU"
-   bottom: "conv2"
-   top: "conv2"
- }

- layer {
-   name: "relu1"
-   type: "ReLU"
-   bottom: "pool1"
-   top: "pool1"
- }

- Memory Reuse
- The input and output of the blob set to the same name, can be a single element of the operation of the relu save storage space

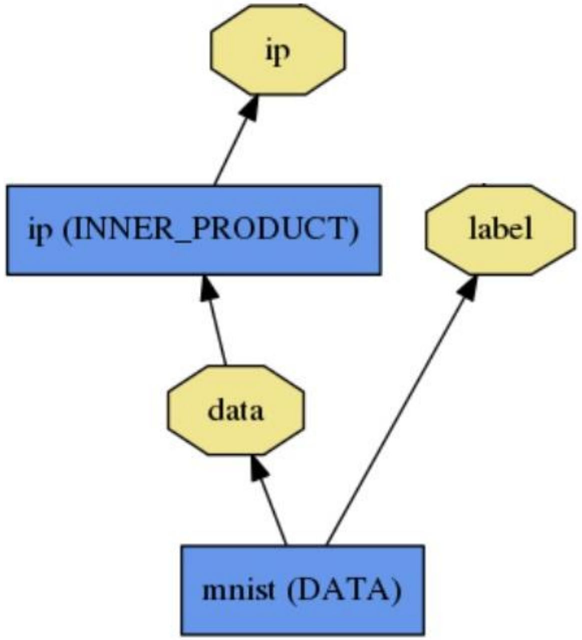
Loss layer(Softmax Layer)

- layer {
- name: "prob"
- type: "Softmax"
- bottom: "ip2"
- top: "prob"
- }

Loss

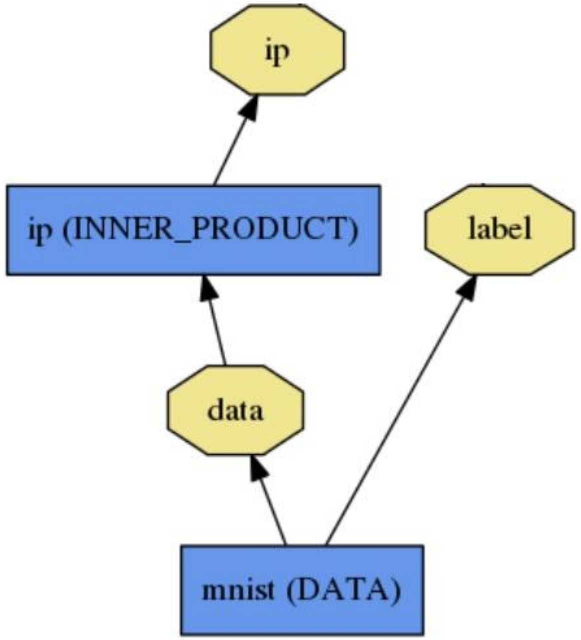
http://caffe.berkeleyvision.org/tutorial/loss.html

What kind of model is this?

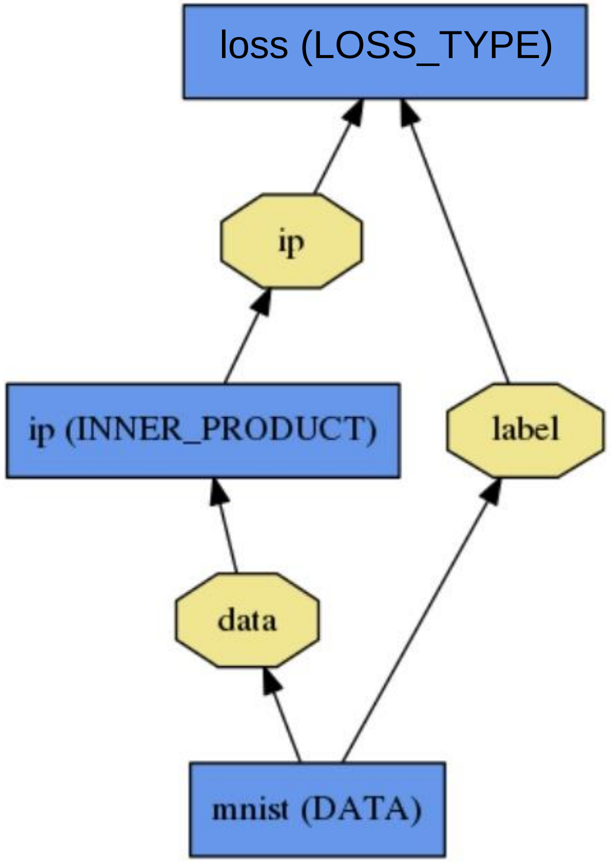


Loss

What kind of model is this?



Define the task by the **loss**.



Classification
SoftmaxWithLoss
HingeLoss

Linear Regression
EuclideanLoss

Attributes / Multiclassifica
SigmoidCrossEntropyLoss

Others…

New Task
NewLoss

The CIFAR-10 dataset

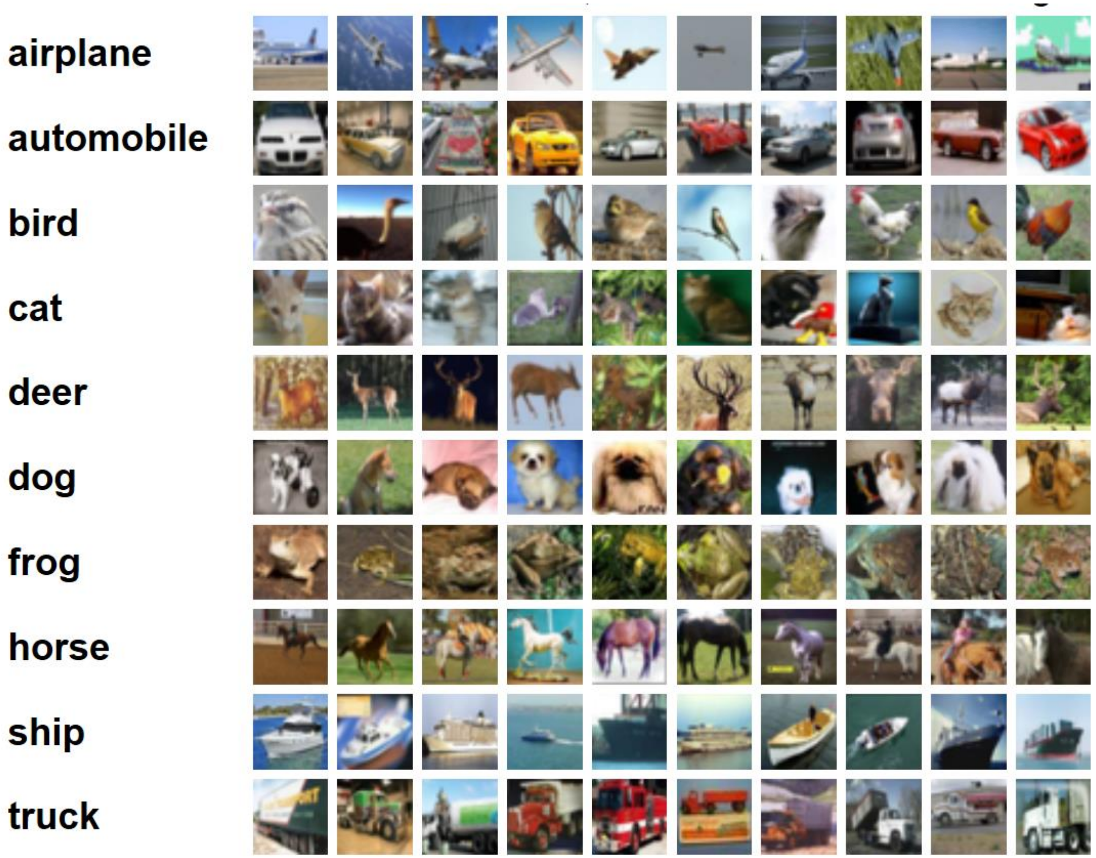
http://www.cs.toronto.edu/~kriz/cifar.html

60000 32x32 colour images in 10 classes,

6000 images per class.

50000 training images

10000 test images.



The CIFAR-100 dataset

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. T

Share a Sip of Brewed Models

demo.caffe.berkeleyvision.org

demo code open-source and bundled

Maximally accurate	Maximally specific
cat	1.80727
domestic cat	1.74727
feline	1.72787
tabby	0.99133
domestic animal	0.78542

LAST SIP

Caffe...

- is fast
- is state-of-the-art
- has tips, recipes, demos, and models
- brings together an active community
- ...all for free and open source

Thank you