Deep Learning:
Tools and Applications

Tools

- Caffe
- Torch
- Theano
- Tensorflow
- Keras
- DIGIT
- ...

Tools

- Caffe
- Torch
- Theano
- Tensorflow
- Keras
- DIGIT
- ...

Datasets

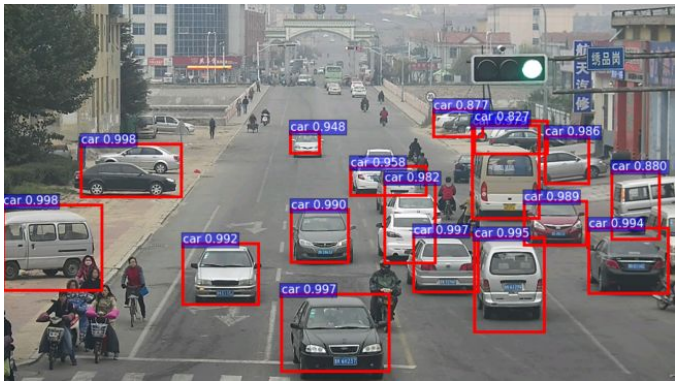
- MNIST
  - Training, validation data: 60000, 10000
  - 32x32(28x28),10類
  - gray scale
- Cifar10
  - Training, validation data: 50000, 10000
  - 32x32, 10類
  - color
- Imagenet
  - Training, validation data: 1281167, 50000
  - 256x256, 1000類
  - color

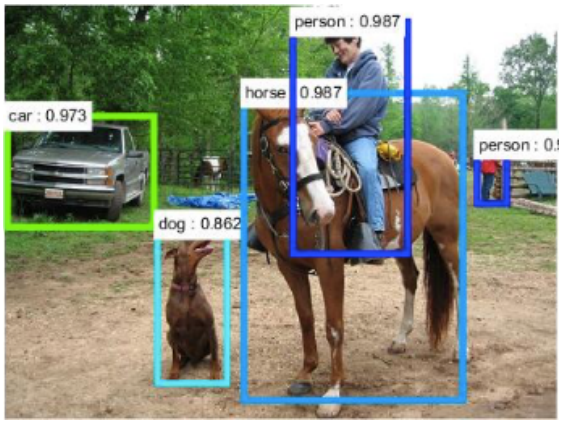
Caffe

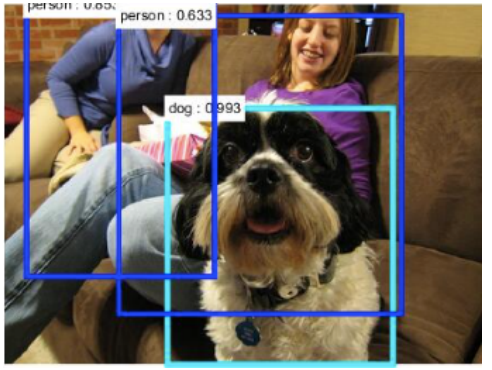
- Windows, OS X and Linux
  - caffe installation: http://caffe.berkeleyvision.org/installation.html
- Python and MATLAB
- Prerequisites
  - BLAS
  - BOOST
  - Protobuf, glog, gflags, hdf5
  - CUDA for GPU mode
    - nvidia cuda-gpus: https://developer.nvidia.com/cuda-gpus
- Optional
  - CuDNN for GPU acceleration
  - OpenCV

Caffe

- Applications
  - Classification
  - Localization (faster-rcnn)





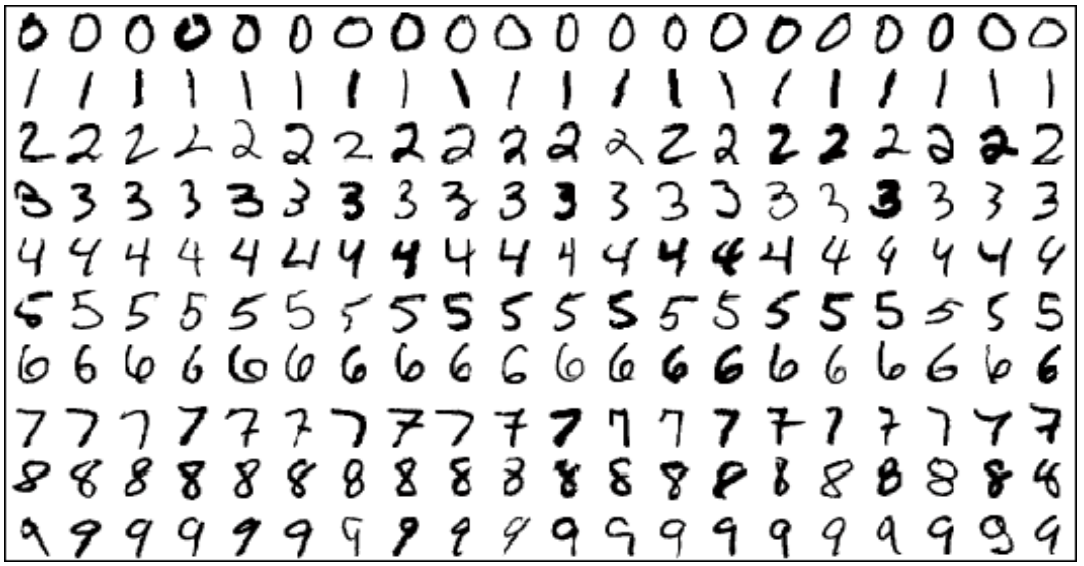


  - Pixel level classification and segmentation
  - Sequence learning (RNN, LSTM)
    - Audio, motion…
  - Transfer learning

Caffe: Classification

- MNIST

```
cd $CAFFE_ROOT
./data/mnist/get_mnist.sh
./examples/mnist/create_mnist.sh
./examples/mnist/train_lenet.sh
```



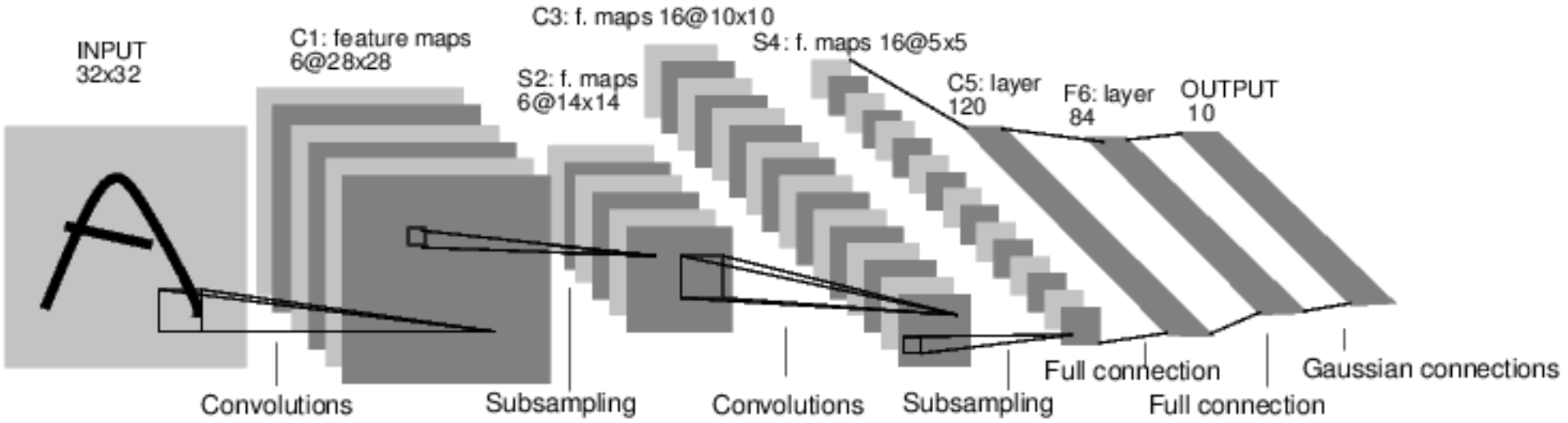
```
.cpp:404]     Test net output #0: accuracy = 0.9909
.cpp:404]     Test net output #1: loss = 0.0294163 (* 1 = 0.0294163 loss)
.cpp:322] Optimization Done.
cpp:254] Optimization Done.
```

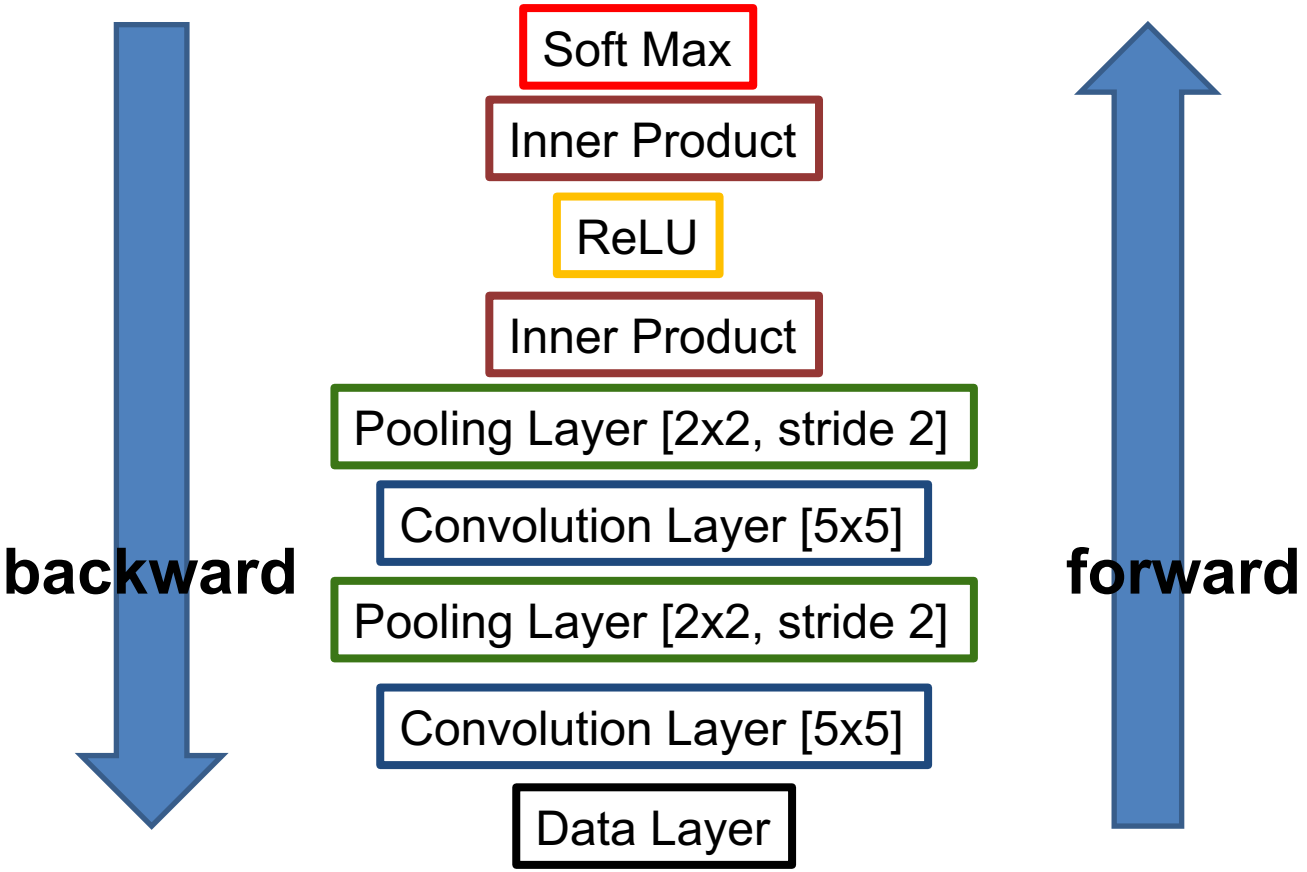
Training data: 60000
Validation data: 10000

MNIST

- Linear classifier
  8%~12% error
- K-nearest-neighbor
  1.x%~5% error
- SVM
  0.6%~1.4% error
- Convolution Nets
  0.3%~1% error

LeNet-5


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections


Soft Max
Inner Product
ReLU
Inner Product
Pooling Layer [2x2, stride 2]
Convolution Layer [5x5]
Pooling Layer [2x2, stride 2]
Convolution Layer [5x5]
Data Layer
backward
forward

LeNet-5: 定義TRAIN data layer

```
1  name: "LeNet"
2  layer {
3    name: "mnist"
4    type: "Data"
5    top: "data"
6    top: "label"
7    include {
8      phase: TRAIN
9    }
10   transform_param {
11     scale: 0.00390625
12   }
13   data_param {
14     source: "MNIST_train_lmdb"
15     batch_size: 64
16     backend: LMDB
17   }
18 }
```

定義該層的名稱

data類型

產生data blob

產生label blob

只在TRAIN階段使用

正規化係數到[0,1) , 1. / 256

批次處理的大小

Blob: Caffe 中一種4D的結構

N*C*H*W
(Num*Channels*Height*Width)

LeNet-5: 定義TEST data layer

```
19 layer {
20   name: "mnist"
21   type: "Data"
22   top: "data"
23   top: "label"
24   include {
25     phase: TEST
26   }
27   transform_param {
28     scale: 0.00390625
29   }
30   data_param {
31     source: "MNIST_val_lmdb"
32     batch_size: 100
33     backend: LMDB
34   }
35 }
```

定義該層的名稱

data類型

產生data blob

產生label blob

只在TEST階段使用

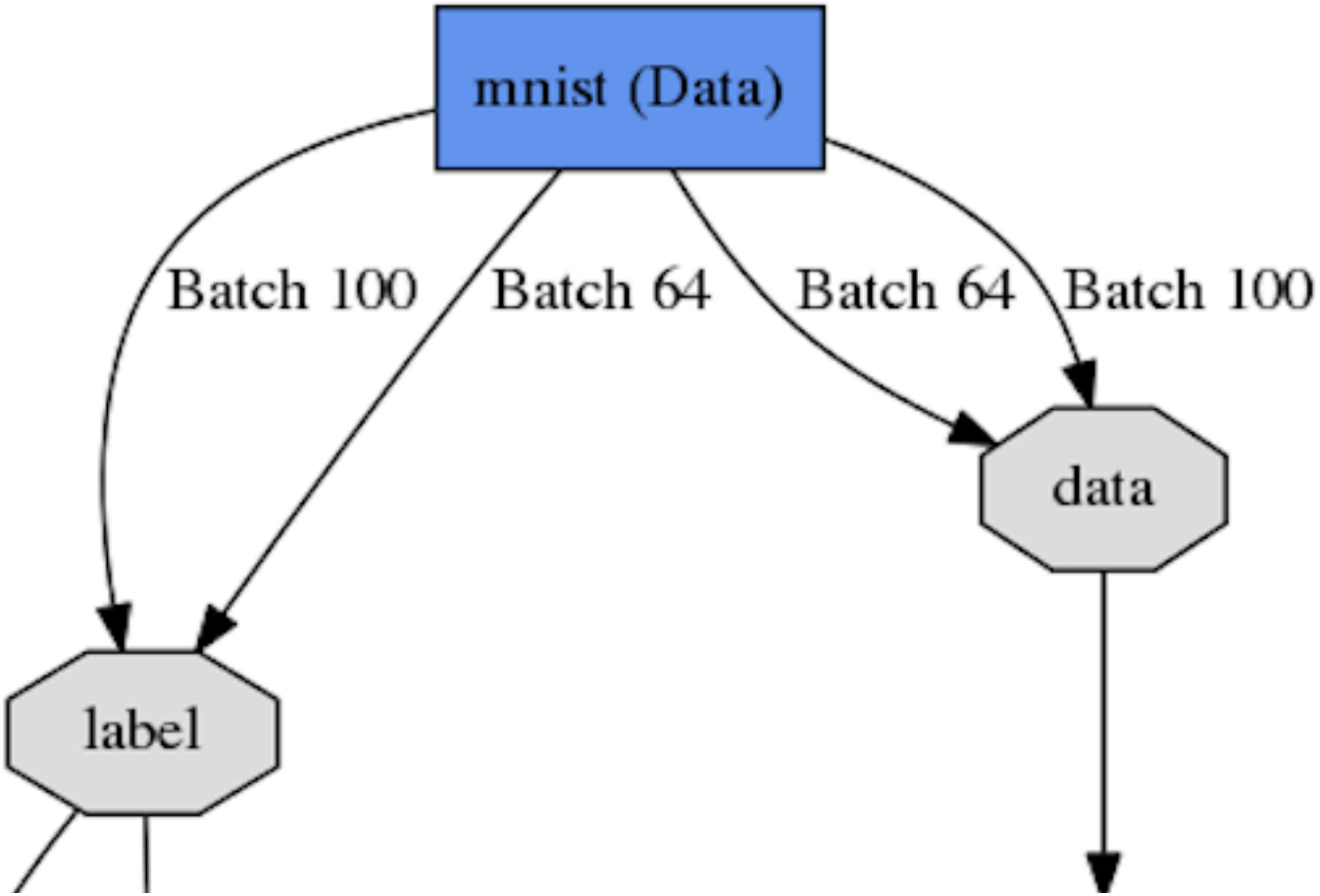
正規化係數到0,1) , 1. / 256

批次處理的大小

Caffe: data type

- Data
  - Dataset 來自於LMDB, LevelDB
- MemoryData
  - Dataset 來自於記憶體
- HDF5Data
  - Dataset 來自於HDF5
- ImageData
  - Dataset 來自於圖片

LeNet-5: Data Layer



batch_size * 1 * 28 * 28

LeNet-5: 定義第一層 convolution layer

```
36 layer {
37   name: "conv1"
38   type: "Convolution"
39   bottom: "data"
40   top: "conv1"
41   param {
42     lr_mult: 1
43   }
44   param {
45     lr_mult: 2
46   }
47   convolution_param {
48     num_output: 20
49     kernel_size: 5
50     stride: 1
51     weight_filler {
52       type: "xavier"
53     }
54     bias_filler {
55       type: "constant"
56     }
57   }
58 }
```

該層的名稱

該層的類型為卷積層

該層使用data layer的data blob

該層生成的data 為 conv1

weight learning rate, lr,
為solver.prototxt中base_lr * lr_mult

bias learning rate,
為solver.prototxt中base_lr * lr_mult

產生20個filter

卷積核大小為5x5

卷積核步幅為1

根據輸入和輸出自動初始化權重比例

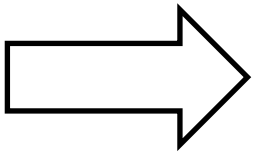
將bias初始化, 默認值為0

Caffe: Convolution Layer

Kernel size: 2x2, stride: 2

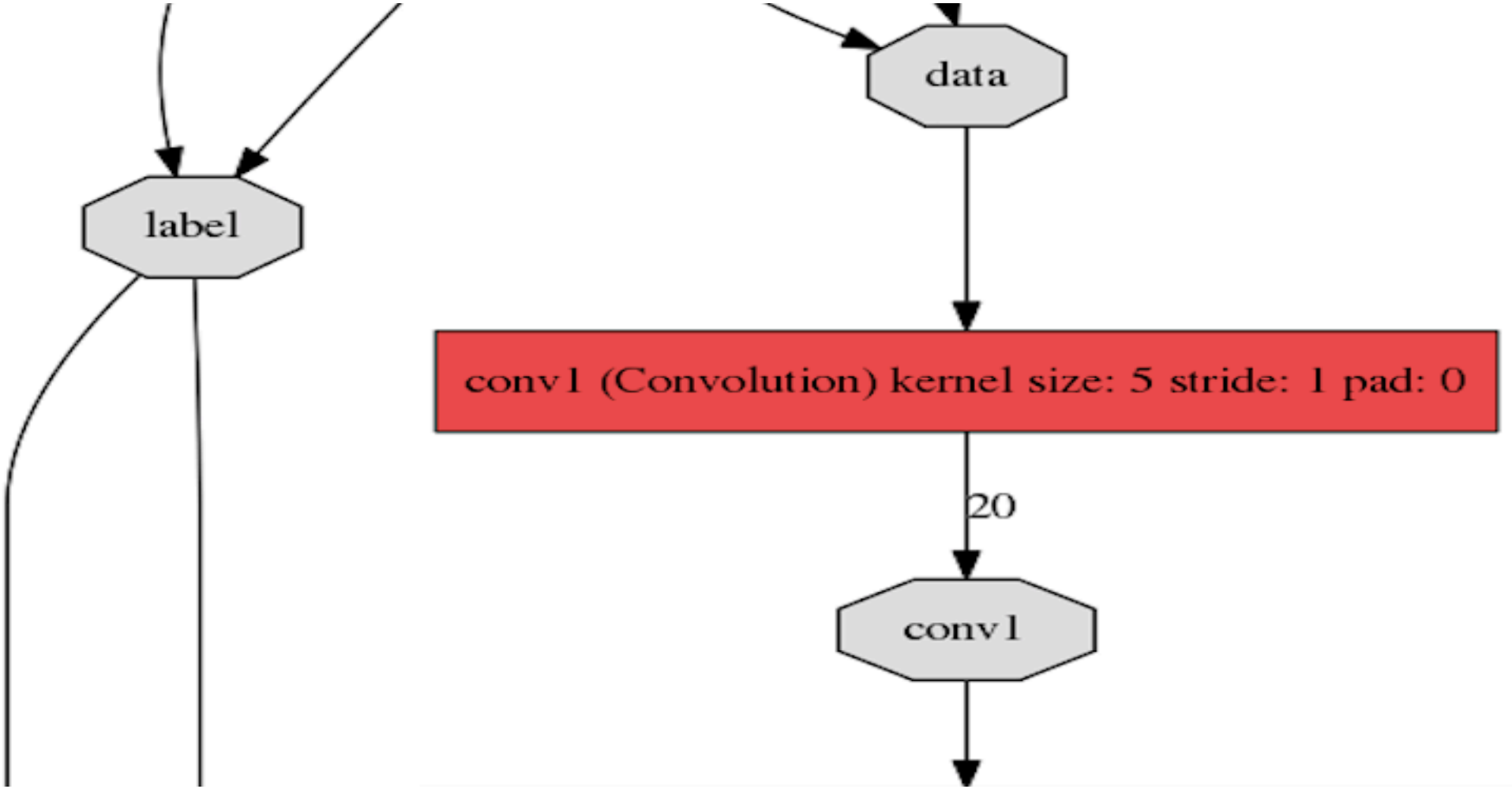
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

1	0	1
0	1	0
1	0	1



4	3	4
2	4	3
2	3	4

LeNet-5: Convolution Layer



conv1: batch_size * 1 * 28 * 28 -> batch_size * 20 * 24 * 24

N * ci * hi * wi -> N * co * ho * wo

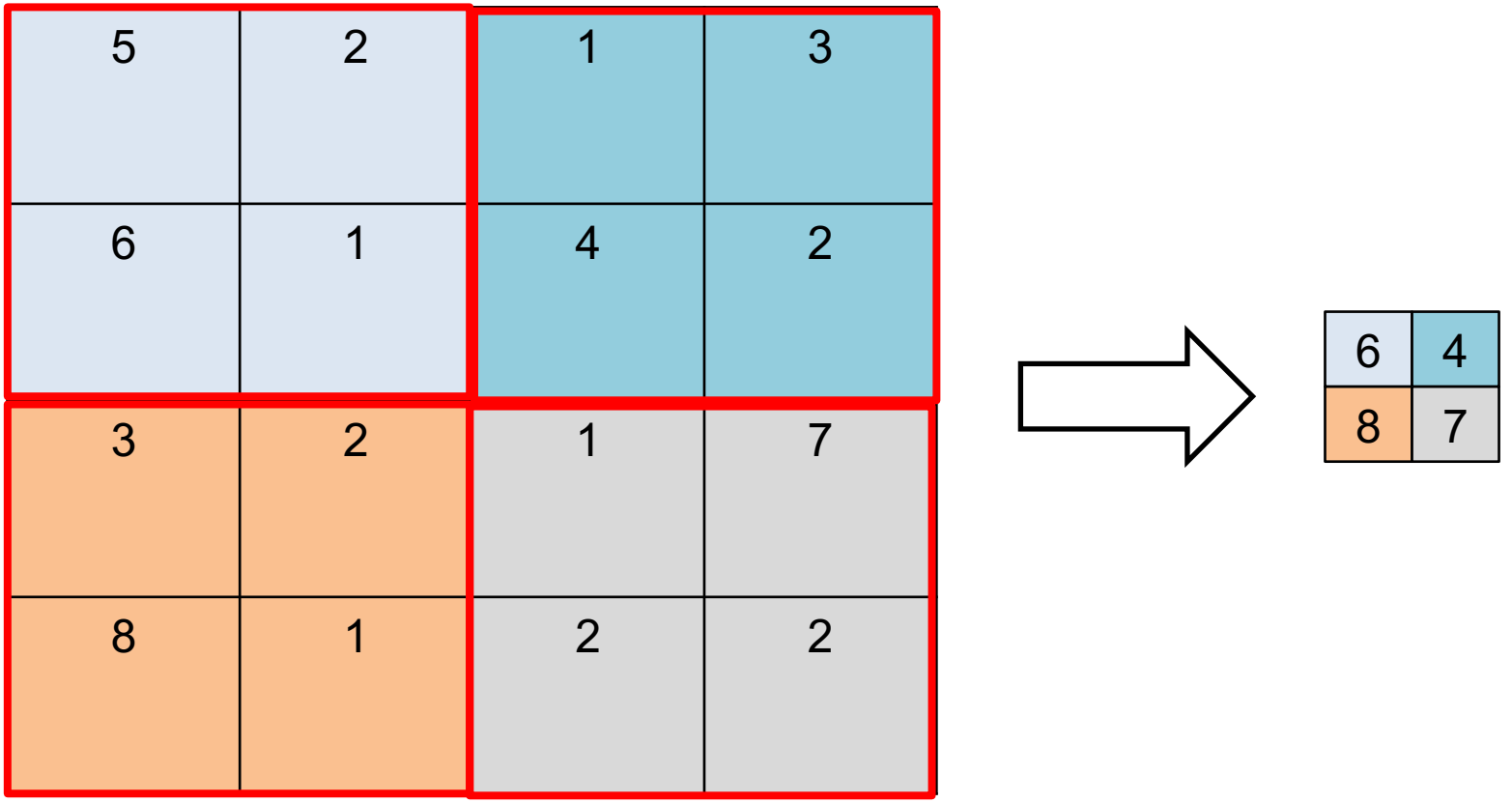
ho = (hi + 2 * padh – kernel_h) / stride_h +1, 同wo

LeNet-5:定義第一層 Pooling layer

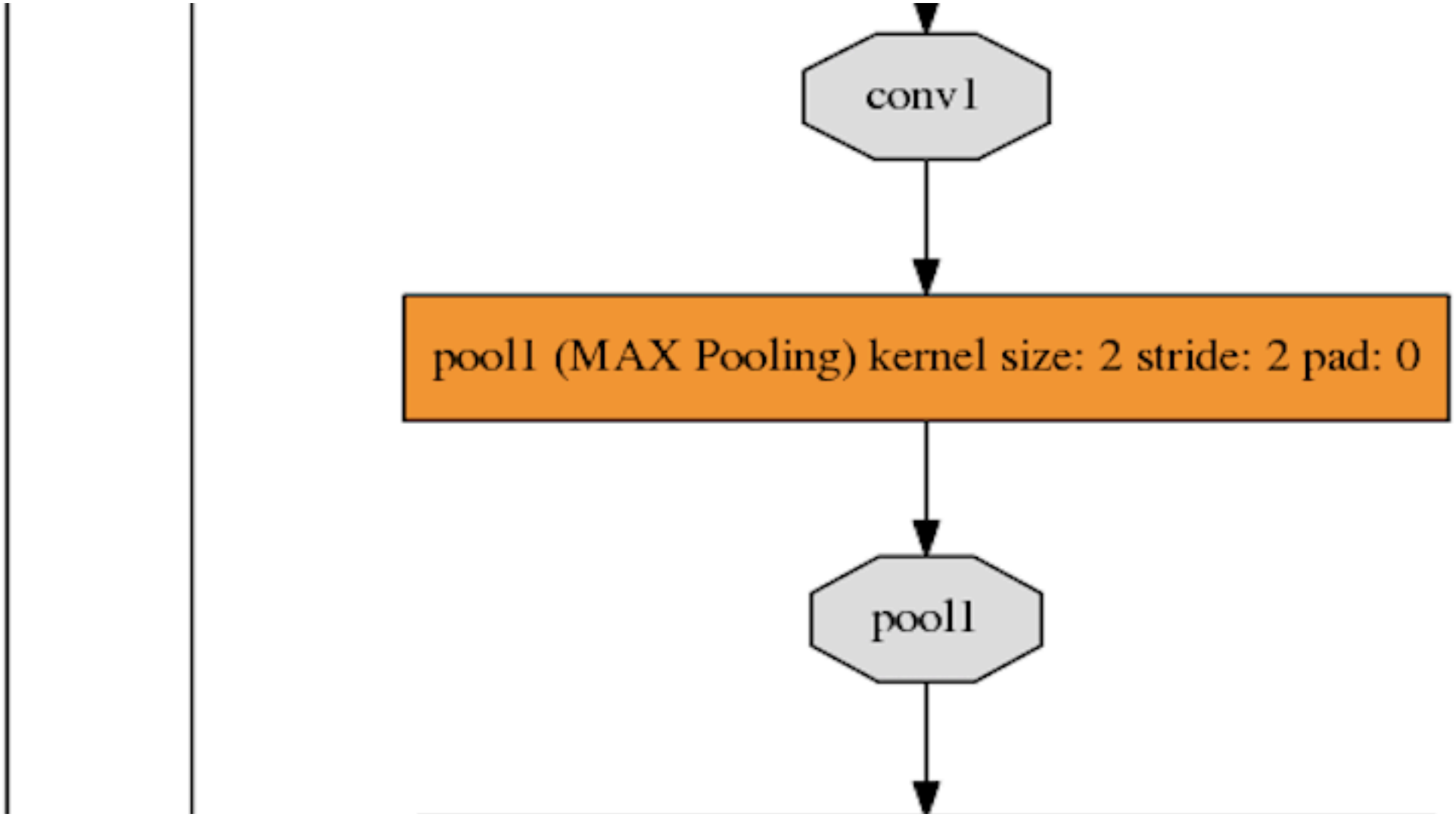
```
59  layer {
60    name: "pool1" 該層的名稱
61    type: "Pooling" 該層的類型為池化層
62    bottom: "conv1" 該層使用conv 1
63    top: "pool1" 該層生成的data 為 pool1
64    pooling_param {
65      pool: MAX 使用MAX pooling
66      kernel_size: 2 Pooling kernel為2x2
67      stride: 2 步幅為2, 即不重疊
68    }
69  }
```

Caffe: MAX Pooling

Kernel size: 2x2, stride: 2



LeNet-5: Pooling Layer



pool1: batch_size * 20 * 24 * 24 -> batch_size * 20 * 12 * 12

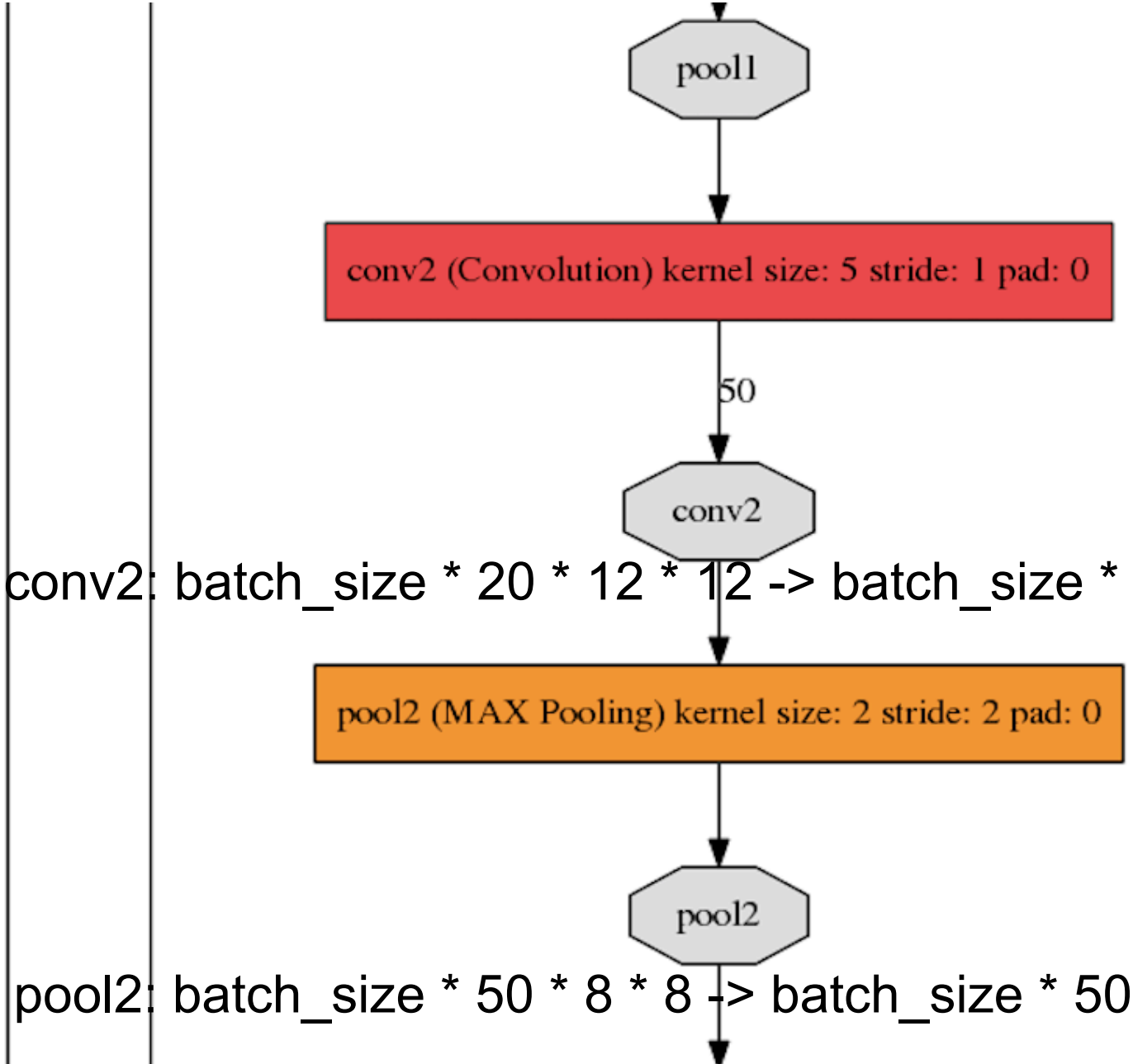
LeNet-5: 定義第二層 convolution layer

```
70 layer {
71   name: "conv2"
72   type: "Convolution"
73   bottom: "pool1"
74   top: "conv2"
75   param {
76     lr_mult: 1
77   }
78   param {
79     lr_mult: 2
80   }
81   convolution_param {
82     num_output: 50
83     kernel_size: 5
84     stride: 1
85     weight_filler {
86       type: "xavier"
87     }
88     bias_filler {
89       type: "constant"
90     }
91   }
92 }
```

LeNet-5:定義第二層 Pooling layer

```
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

LeNet-5: conv2+pool2



conv2: batch_size * 20 * 12 * 12 -> batch_size * 50 * 8 * 8

pool2: batch_size * 50 * 8 * 8 -> batch_size * 50 * 4 * 4

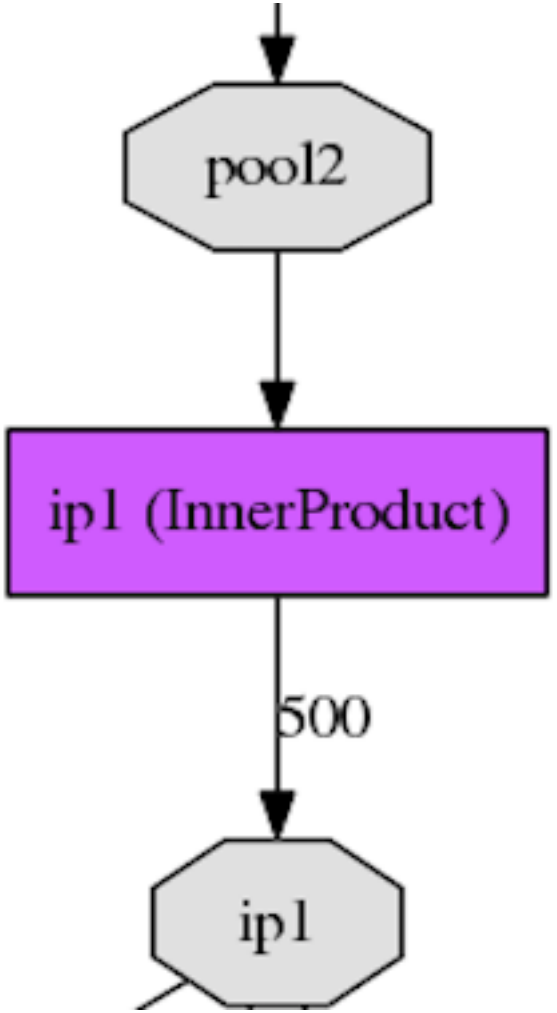
LeNet-5:定義第一層全連接層

```
104 layer {
105   name: "ip1"
106   type: "InnerProduct"
107   bottom: "pool2"
108   top: "ip1"
109   param {
110     lr_mult: 1
111   }
112   param {
113     lr_mult: 2
114   }
115   inner_product_param {
116     num_output: 500
117     weight_filler {
118       type: "xavier"
119     }
120     bias_filler {
121       type: "constant"
122     }
123   }
124 }
```

該層的類型為全連接層

輸出500個

LeNet-5: Inner Product, ip1



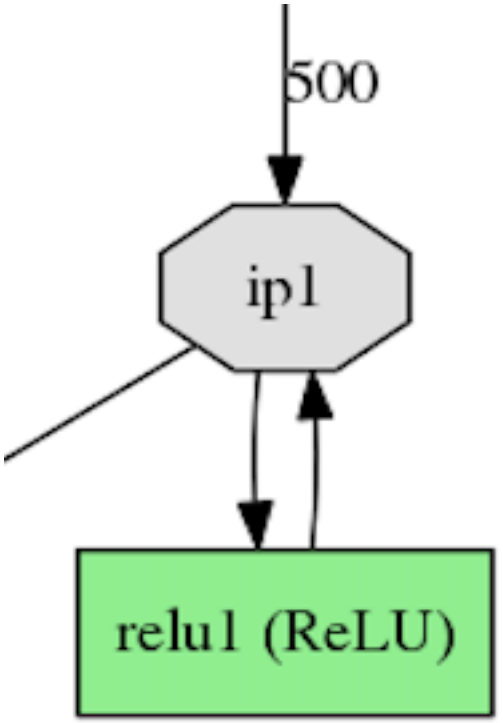
ip1: batch_size * 50 * 4 * 4 -> batch_size * 500 * 1 * 1
全連接是將c*h*w 轉成1D vector

LeNet-5:定義第一層ReLU

```
125 layer {
126   name: "relu1"
127   type: "ReLU"
128   bottom: "ip1"
129   top: "ip1"
130 }
```

該層的類型為ReLU

LeNet-5: ReLU

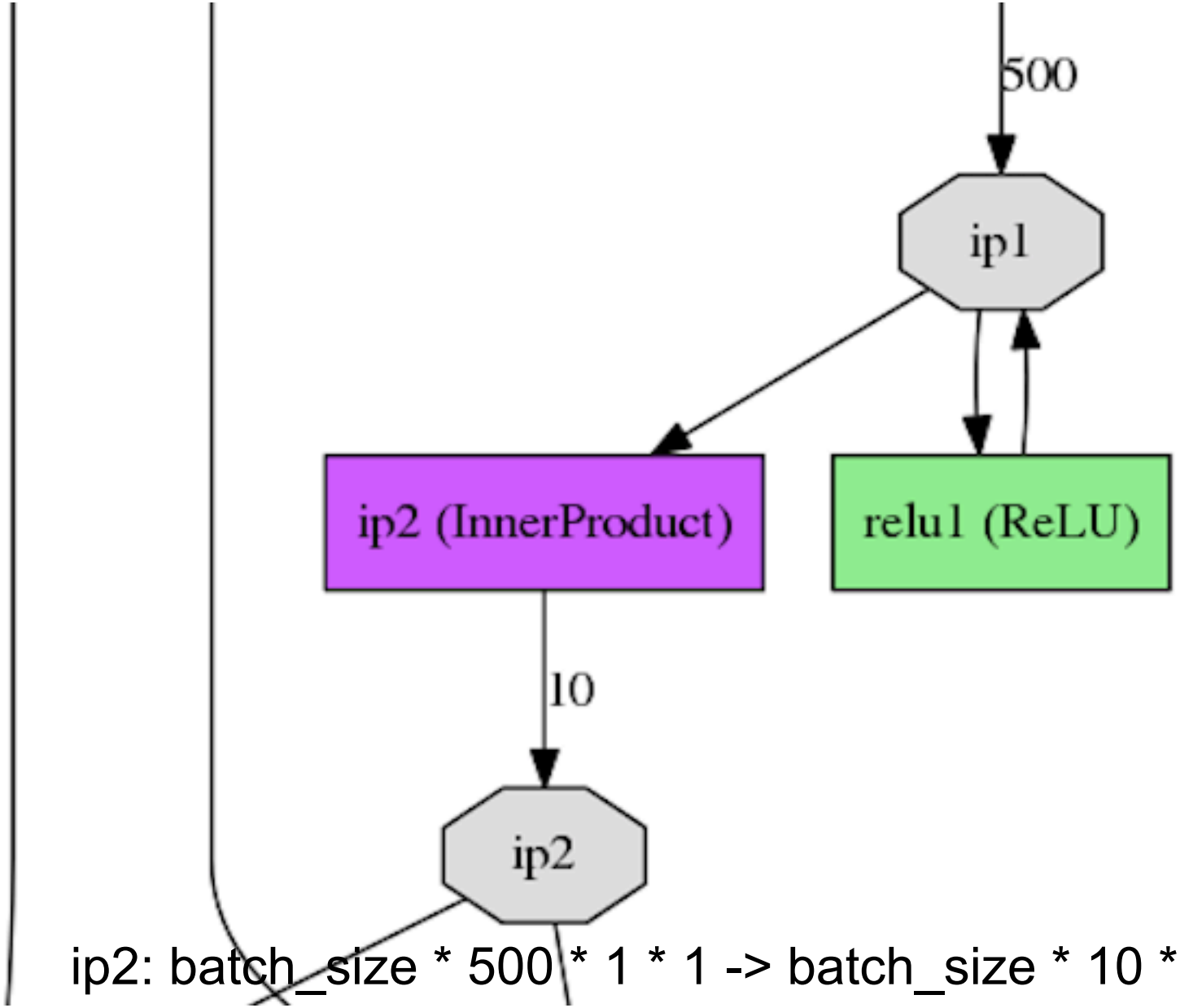


relu1: batch_size * 500 * 1 * 1 -> batch_size * 500 * 1 * 1

LeNet-5:定義第二層全連接層

```
131 layer {
132   name: "ip2"
133   type: "InnerProduct"
134   bottom: "ip1"
135   top: "ip2"
136   param {
137     lr_mult: 1
138   }
139   param {
140     lr_mult: 2
141   }
142   inner_product_param {
143     num_output: 10   輸出10個, 對應類別數, 即0-9 十個數字
144     weight_filler {
145       type: "xavier"
146     }
147     bias_filler {
148       type: "constant"
149     }
150   }
151 }
```

LeNet-5: ip2



ip2: batch_size * 500 * 1 * 1 -> batch_size * 10 * 1 * 1

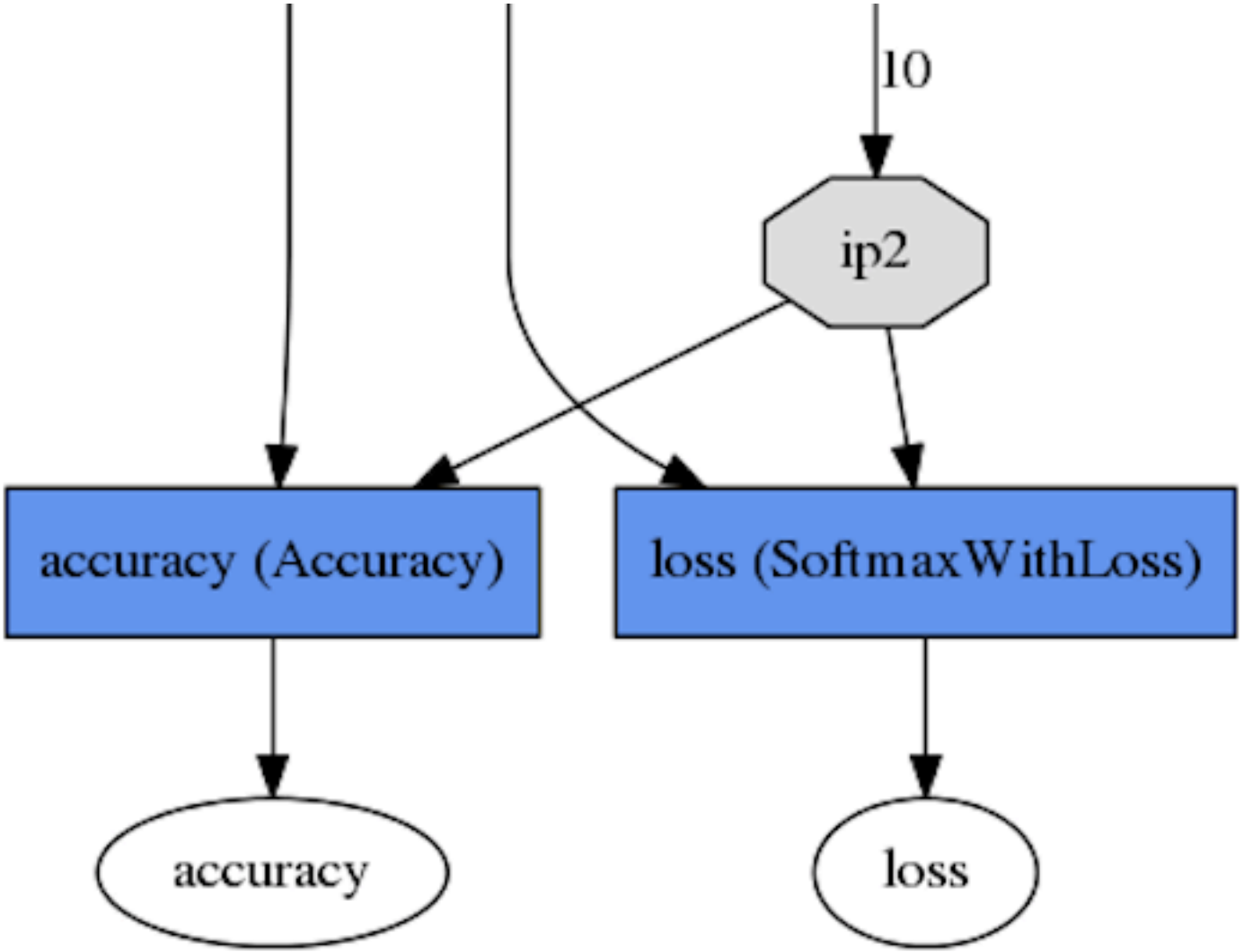
LeNet-5:定義準確率結果層

```
152 layer {
153   name: "accuracy"
154   type: "Accuracy"
155   bottom: "ip2"
156   bottom: "label"
157   top: "accuracy"
158   include {
159     phase: TEST
160   }
161 }
```

LeNet-5:定義損失函數層

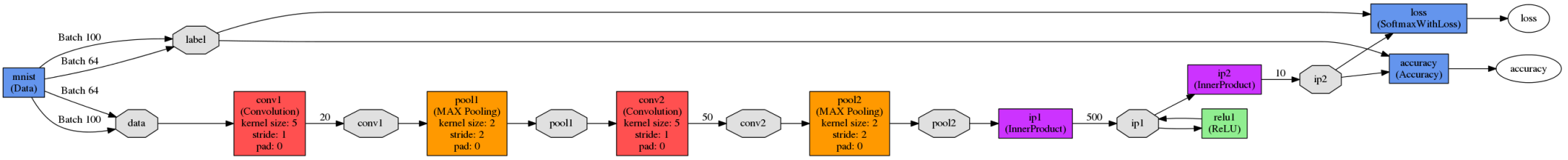
```
159 layer {
160   name: "loss"
161   type: "SoftmaxWithLoss"
162   bottom: "ip2"
163   bottom: "label"
164   top: "loss"
165 }
```

LeNet-5: accuracy + loss



loss: batch_size * 10 * 1 * 1 -> batch_size * 10 * 1 * 1

LeNet-5


mnist
(Data)
Batch 100
Batch 64
Batch 64
Batch 100
label
data
conv1
(Convolution)
kernel size: 5
stride: 1
pad: 0
20
conv1
pool1
(MAX Pooling)
kernel size: 2
stride: 2
pad: 0
pool1
conv2
(Convolution)
kernel size: 5
stride: 1
pad: 0
50
conv2
pool2
(MAX Pooling)
kernel size: 2
stride: 2
pad: 0
pool2
ip1
(InnerProduct)
500
ip1
ip2
(InnerProduct)
relu1
(ReLU)
10
ip2
loss
(SoftmaxWithLoss)
loss
accuracy
(Accuracy)
accuracy

Caffe Net With Python

```
from caffe import layers as L, params as P

def lenet(lmdb, batch_size):
    # our version of LeNet: a series of linear and simple nonlinear transformations
    n = caffe.NetSpec()

    n.data, n.label = L.Data(batch_size=batch_size, backend=P.Data.LMDB, source=lmdb,
                             transform_param=dict(scale=1./255), ntop=2)

    n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20, weight_filler=dict(type='xavier'))
    n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50, weight_filler=dict(type='xavier'))
    n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.fc1 =   L.InnerProduct(n.pool2, num_output=500, weight_filler=dict(type='xavier'))
    n.relu1 = L.ReLU(n.fc1, in_place=True)
    n.score = L.InnerProduct(n.relu1, num_output=10, weight_filler=dict(type='xavier'))
    n.loss =  L.SoftmaxWithLoss(n.score, n.label)

    return n.to_proto()

with open('mnist/lenet_auto_train.prototxt', 'w') as f:
    f.write(str(lenet('mnist/mnist_train_lmdb', 64)))

with open('mnist/lenet_auto_test.prototxt', 'w') as f:
    f.write(str(lenet('mnist/mnist_test_lmdb', 100)))
```

Example: http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/01-learning-lenet.ipynb

名詞

- Training data (50%)
  在訓練過程中用來學習的data set
- Validation data (30%)
  在訓練過程中習用來驗証的data set
- Test data (20%)
  用來評估最後的model
- Batch_size
  在一次forward/backward處理的n個data
- Iteration
  一次為一個Batch_size，forward/backward一次
- Epoch
  forward/backward 全部的data
  Ex: dataset共10000張，Batch_size=100
  ⇒1個epoch = 100個iteration

Caffe: Train CNN on your own data

- Dataset: MNIST
  - Training data: 60000/10
  - Validation data: 10000/10
- Create label text
  - format
    - Folder/image_file_name (空格) label
  - Ex:

```
4/30863_4.png 4
4/53134_4.png 4
4/25925_4.png 4
4/6656_4.png 4
4/44694_4.png 4
4/3280_4.png 4
0/35192_0.png 0
0/5167_0.png 0
0/52449_0.png 0
0/712_0.png 0
0/40456_0.png 0
0/849_0.png 0
0/27603_0.png 0
```

```
|---MNIST
    |---train
        |---0
            |--- *.png
        |---1
            |--- *.png
        |--- ...
    |---val
        |---0
            |--- *.png
        |---...
```

- Note:

Label 從0開始

Caffe: Train CNN on your own data

- Create lmdb
  - 用caffe/examples/imagenet的create_imagenet.sh

    cp caffe/examples/imagenet/create_imagenet.sh path/to/your/MNIST
  - 修改
    - EXAMPLE: lmdb要存的位置
    - DATA: label.txt的位置
    - TRAIN_DATA_ROOT: training data的位置
    - VAL_DATA_ROOT: validation data的位置
  - EX:

```
6   EXAMPLE=/path/to/MNIST
7   DATA=/path/to//MNIST
8   TOOLS=build/tools
9
10  TRAIN_DATA_ROOT=/path/to/MNIST/train/
11  VAL_DATA_ROOT=/path/to/MNIST/val/
```

```
|--MNIST
  |--train
    |--0
      |--*.png
    |--1
      |--*.png
    |-- ...
  |--val
    |--0
      |--*.png
    |--1
      |--*.png
    |-- ...
  |--training_label.txt
  |--val_label.txt
```

Caffe: Train CNN on your own data

- RESIZE:

```
15  RESIZE=true
16  if $RESIZE; then
17    RESIZE_HEIGHT=32
18    RESIZE_WIDTH=32
19  else
20    RESIZE_HEIGHT=0
21    RESIZE_WIDTH=0
```

Resize 影像成 32x32

```
40 GLOG_logtostderr=1 $TOOLS/convert_imageset \
41      --resize_height=$RESIZE_HEIGHT \
42      --resize_width=$RESIZE_WIDTH \
43      --shuffle \
44      --gray \
45      $TRAIN_DATA_ROOT \
46      $DATA/train.txt \
47      $EXAMPLE/ilsvrc12_train_lmdb
```

轉為灰階影像
修改txt的名稱
修改lmdb的名稱

```
51  GLOG_logtostderr=1 $TOOLS/convert_imageset \
52      --resize_height=$RESIZE_HEIGHT \
53      --resize_width=$RESIZE_WIDTH \
54      --shuffle \
55      --gray \
56      $VAL_DATA_ROOT \
57      $DATA/val.txt \
58      $EXAMPLE/ilsvrc12_val_lmdb
```

轉為灰階影像
修改txt的名稱
修改lmdb的名稱

Caffe: Train CNN on your own data

- Now create lmdb
  - ./create_imagenet.sh

```
|--MNIST
   |--train
      |--0
         |--*.png
      |--1
         |--*.png
      |-- ...
   |--val
      |--0
         |--*.png
      |--1
         |--*.png
      |-- ...
   |--training_label.txt
   |--val_label.txt
   |--train_lmdb
   |--val_lmdb
```

- 修改train_test.prototxt
  - 複製lenet_train_test.prototxt
    - cp caffe/examples/mnist/lenet_train_test.prototxt

```
1   name: "LeNet"
2   layer {
3     name: "mnist"
4     type: "Data"
5     top: "data"
6     top: "label"
7     include {
8       phase: TRAIN
9     }
10    transform_param {
11      scale: 0.00390625
12    }
13    data_param {
14      source: "path/to/your/MNIST/train_lmdb"
15      batch_size: 64
16      backend: LMDB
17    }
18  }
```

Normalize 成 0 ~1 (1. /255)

修改路徑到你的train_lmdb

Caffe: Train CNN on your own data

```
19▼ layer {
20     name: "mnist"
21     type: "Data"
22     top: "data"
23     top: "label"
24     include {
25       phase: TEST
26     }
27     transform_param {
28       scale: 0.00390625
29     }
30▼    data_param {
31       source: "path/to/your/MNIST/val_lmdb"
32       batch_size: 100
33       backend: LMDB
34     }
35   }
```

修改路徑到到你的val_lmdb

```
142      inner_product_param {
143        num_output: 10
144        weight_filler {
145          type: "xavier"
146        }
147        bias_filler {
148          type: "constant"
149        }
150      }
151    }
```

要訓練的類別數

Caffe: Train CNN on your own data

```
152   layer {
153     name: "accuracy"
154     type: "Accuracy"
155     bottom: "ip2"
156     bottom: "label"
157     top: "accuracy"
158     include {
159       phase: TEST
160     }
```

只在TEST 階段計算準確率

– 修改solver.prototxt

- cp caffe/examples/mnist/lenet_solver.prototxt

```
1  # The train/test net protocol buffer definition
2  net: "examples/mnist/lenet_train_test.prototxt"
3  # test_iter specifies how many forward passes the test should carry out.
4  # In the case of MNIST, we have test batch size 100 and 100 test iterations,
5  # covering the full 10,000 testing images.
6  test_iter: 100
7  # Carry out testing every 500 training iterations.
8  test_interval: 500
```

Path/to/your/train_test.prototxt

test_iter = total number of validation data / test_batch_size

Caffe: Train CNN on your own data

```
10  base_lr: 0.01
11  momentum: 0.9
12  weight_decay: 0.0005
13  # The learning rate policy
14  lr_policy: "inv"
15  gamma: 0.0001
16  power: 0.75
17  # Display every 100 iterations
18  display: 100
19  # The maximum number of iterations
20  max_iter: 10000
21  # snapshot intermediate results
22  snapshot: 5000
23  snapshot_prefix: "examples/mnist/lenet"
24  # solver mode: CPU or GPU
25  solver_mode: GPU
```


Path/to/store/your/caffemodel

- 開始training

  cd CAFFE_ROOT/build/tools

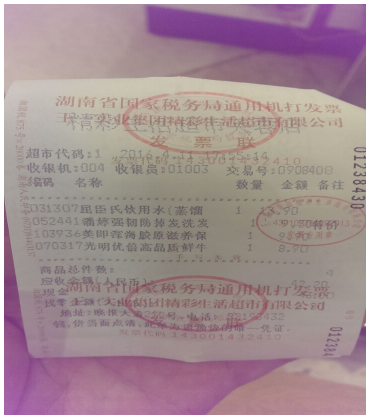
  ./caffe train -solver=/path/to/your/solver.prototxt 2>&1 | tee /path/to/store/training_log.log

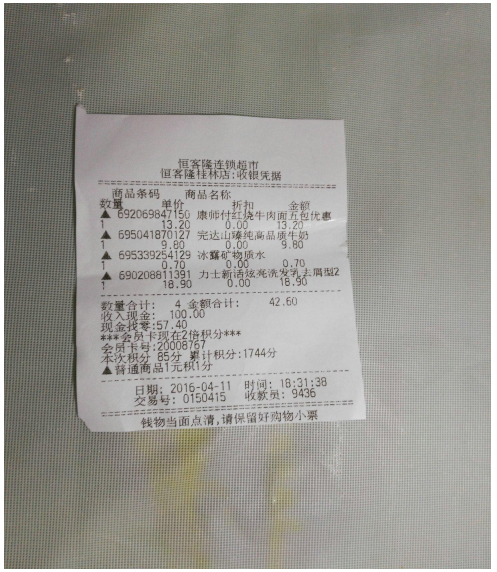
Example: 有沒有小票

- 商品圖:

Example: 有沒有小票

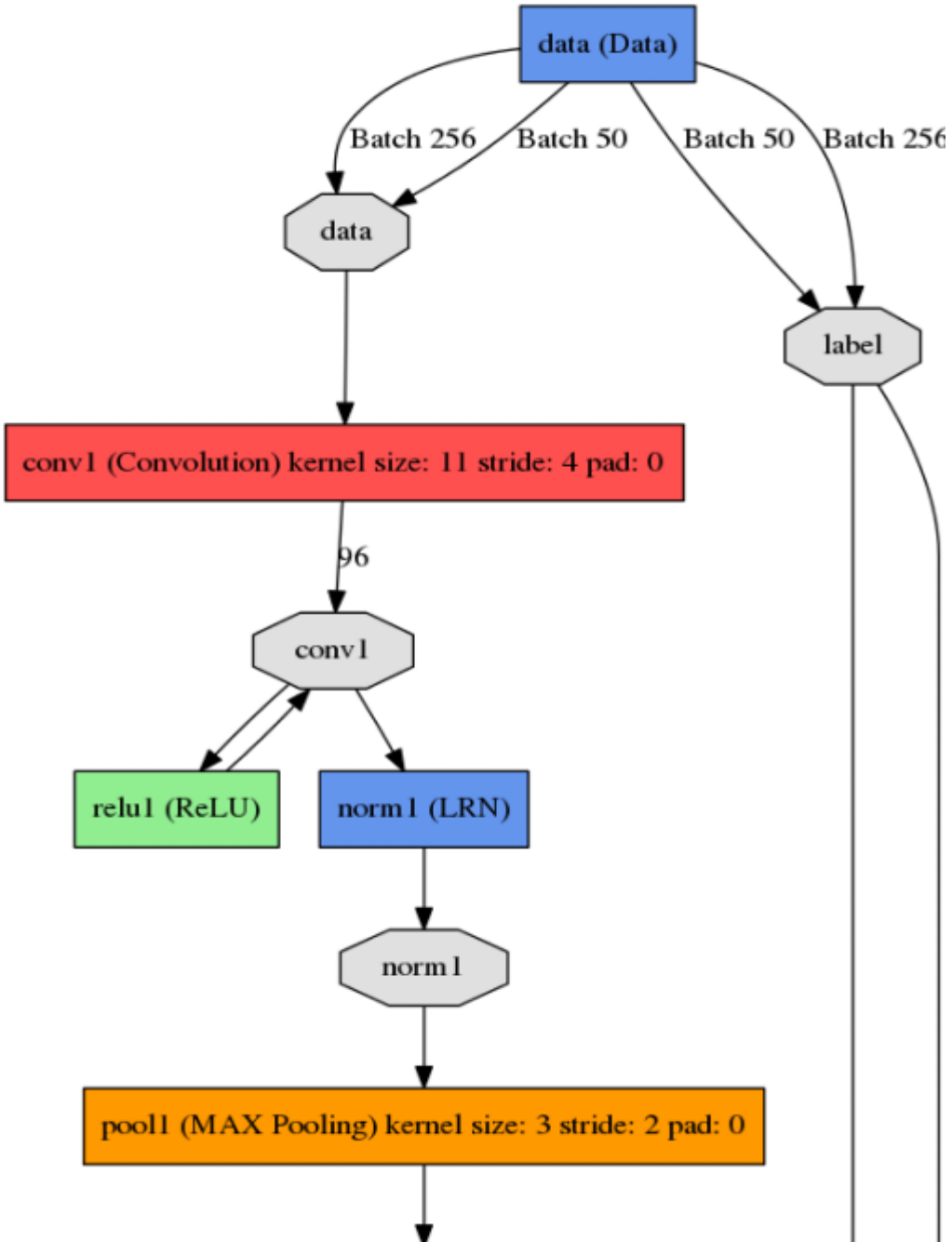
- 小票圖:

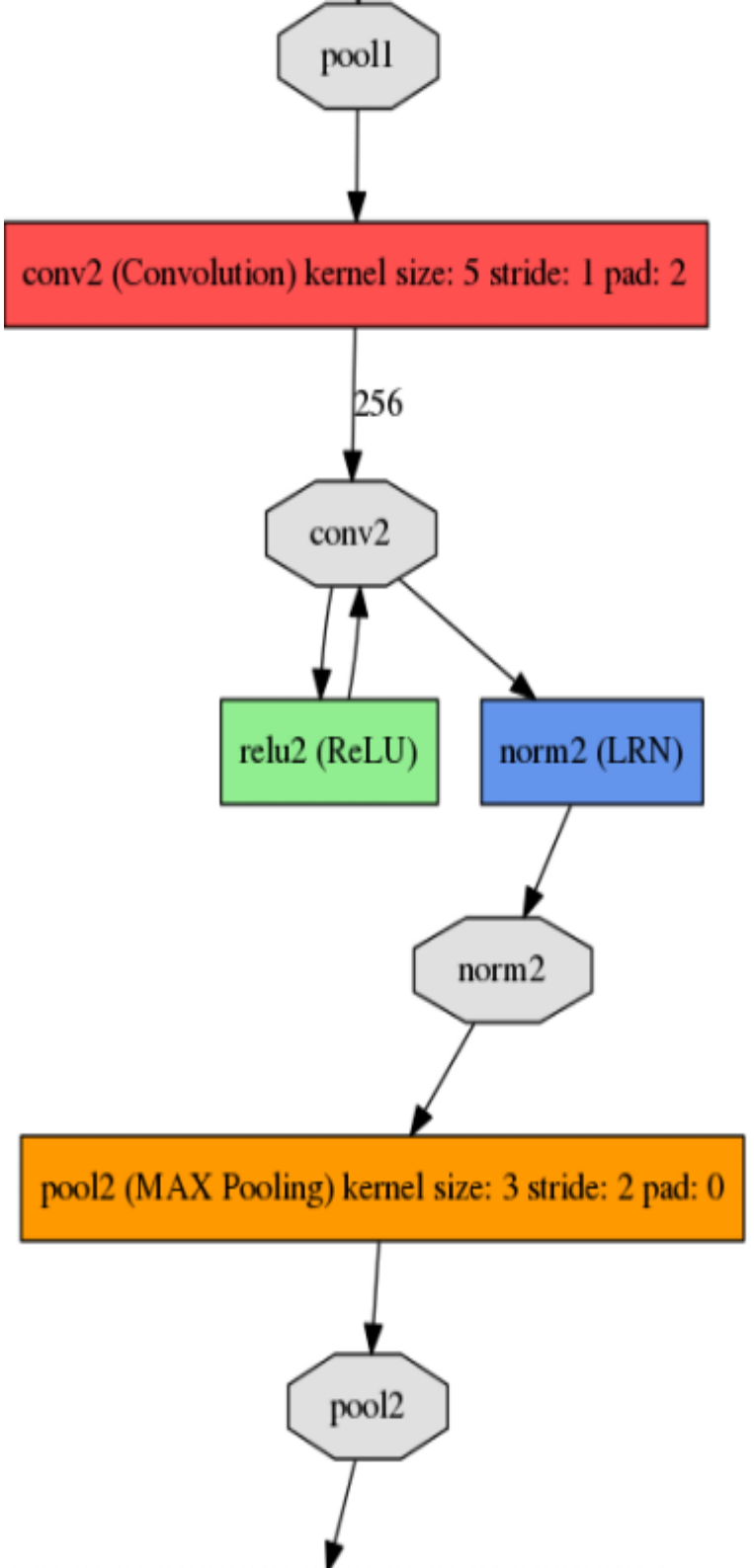


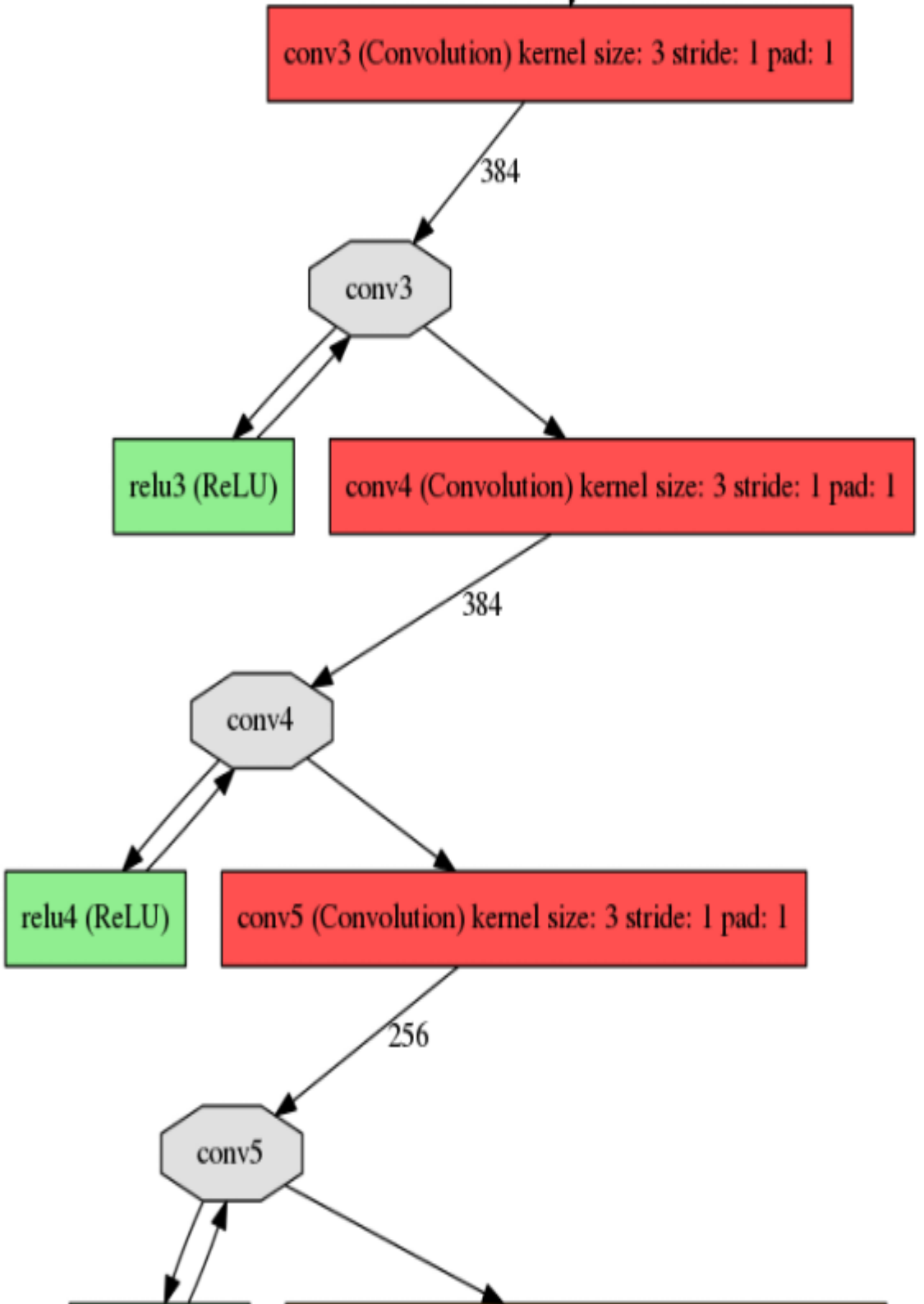


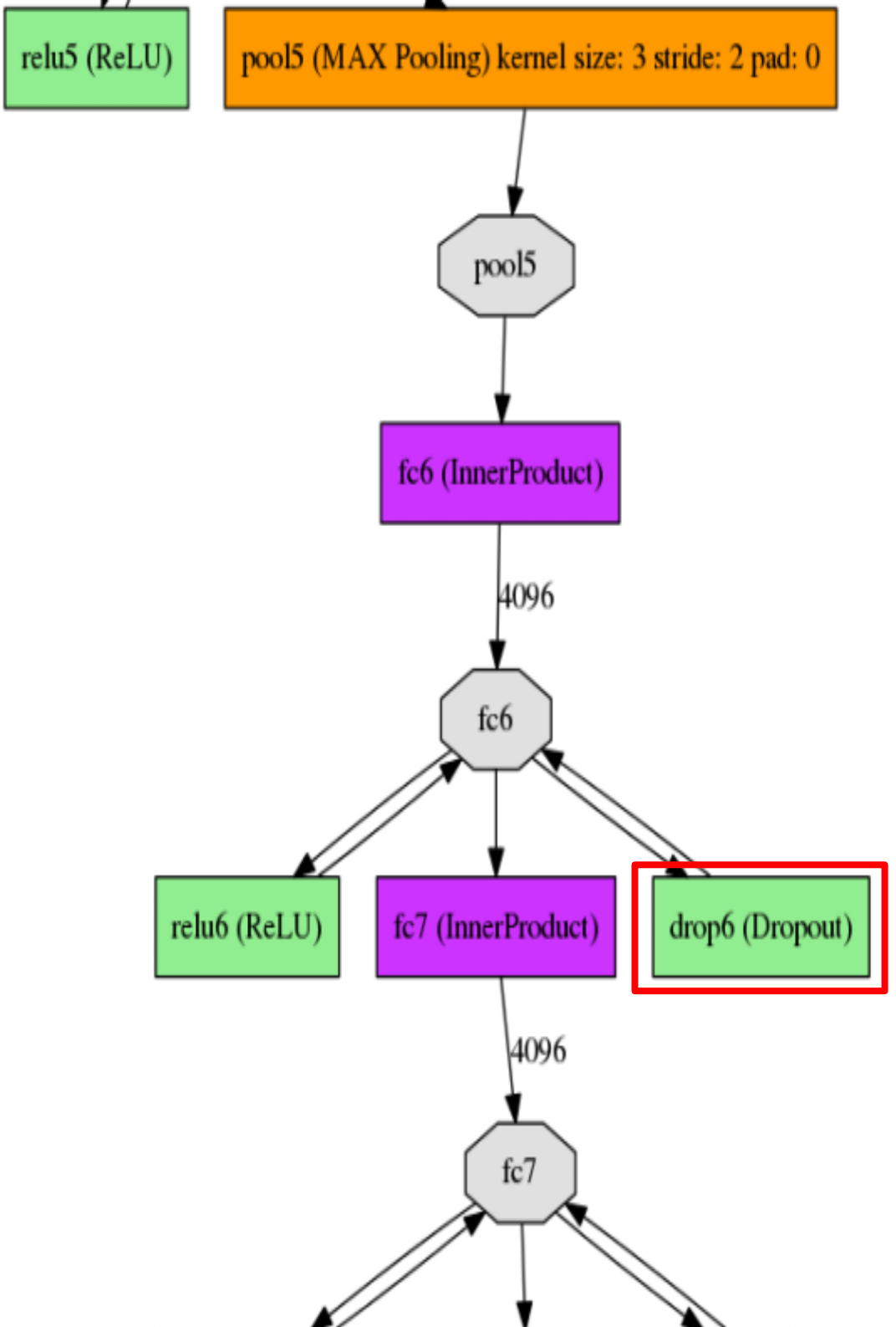
Example: 有沒有小票

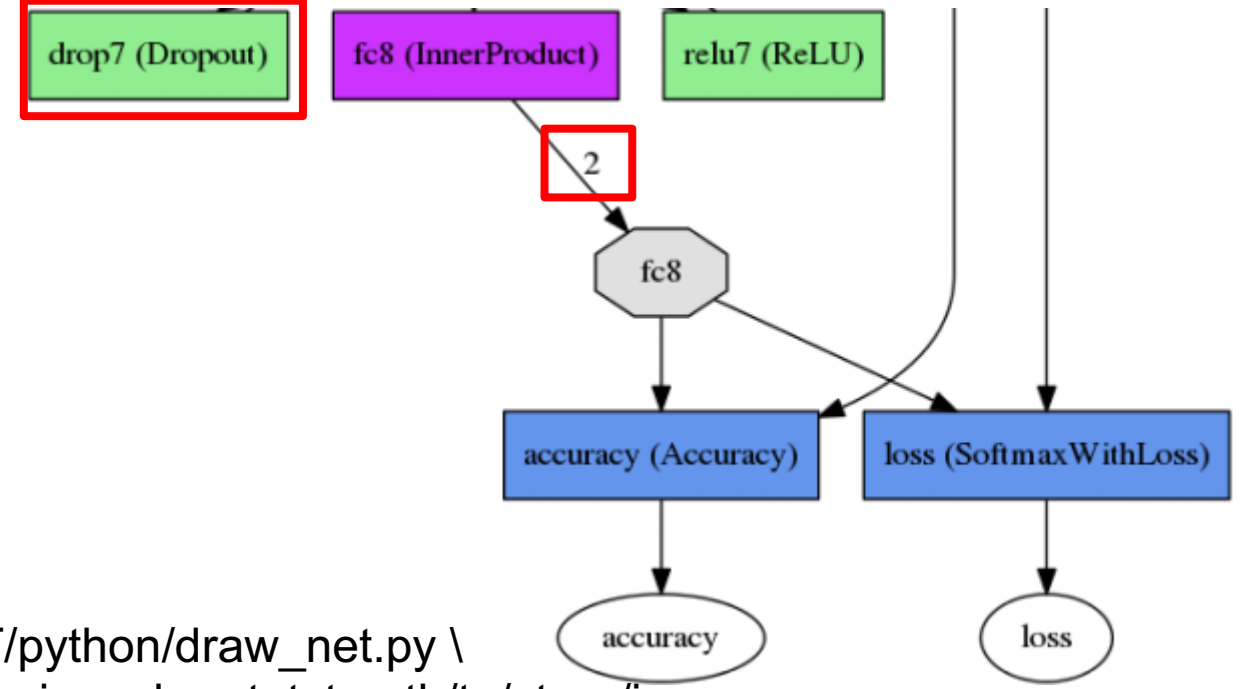
- AlexNet



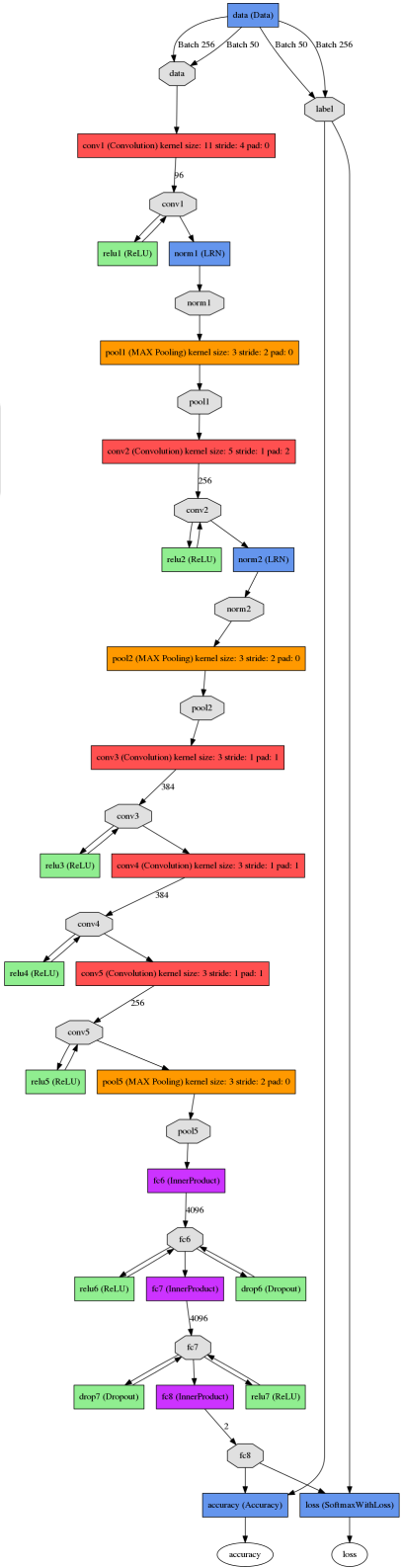

pool1
conv2 (Convolution) kernel size: 5 stride: 1 pad: 2
256
conv2
relu2 (ReLU)
norm2 (LRN)
norm2
pool2 (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool2


conv3 (Convolution) kernel size: 3 stride: 1 pad: 1
384
conv3
relu3 (ReLU)
conv4 (Convolution) kernel size: 3 stride: 1 pad: 1
384
conv4
relu4 (ReLU)
conv5 (Convolution) kernel size: 3 stride: 1 pad: 1
256
conv5

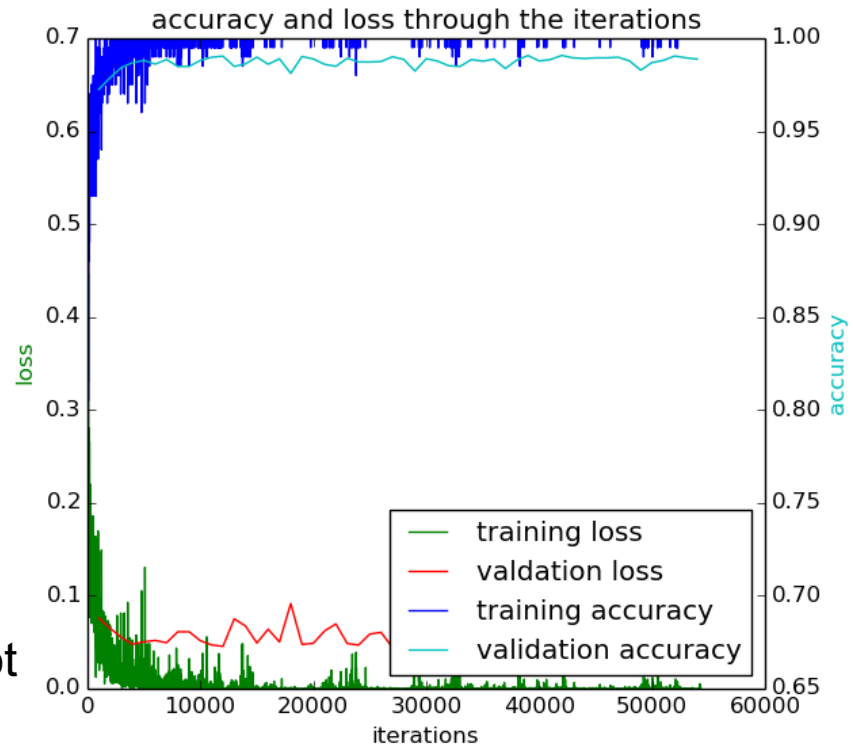

relu5 (ReLU)
pool5 (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool5
fc6 (InnerProduct)
4096
fc6
relu6 (ReLU)
fc7 (InnerProduct)
drop6 (Dropout)
4096
fc7




```
python CAFFE_ROOT/python/draw_net.py \
    path/to/your/train_val.prototxt path/to/store/image
```



Parse log -> gnu plot



Example: 有沒有小票

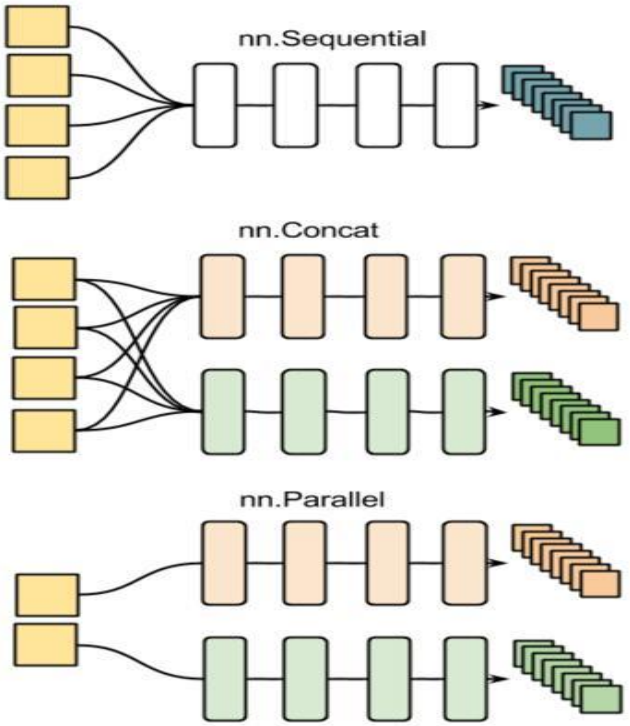
- 小票被認為是商品

Example: 有沒有小票

- 商品被認為是小票

Torch

- Mac OS X, Ubuntu and Windows
- Lua
- Support CUDA, cuDNN
- CNN, RCNN, LSTM
- Container



Torch

```
-- stage 1 : mean suppresion -> filter bank -> squashing -> max pooling
model:add(nn.SpatialConvolutionMM(1, 32, 5, 5))-> 32 x 28 x 28
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(3, 3, 3, 3, 1, 1))-> 32 x 10 x 10
-- stage 2 : mean suppresion -> filter bank -> squashing -> max pooling
model:add(nn.SpatialConvolutionMM(32, 64, 5, 5)) -> 64 x 6 x 6
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2, 2, 2, 2)) -> 64 x 3 x 3
-- stage 3 : standard 2-layer MLP:
model:add(nn.Reshape(64*3*3))
model:add(nn.Linear(64*3*3, 200))
model:add(nn.Tanh())
model:add(nn.Linear(200, #classes))
```

```
model:add(nn.LogSoftMax())
criterion = nn.ClassNLLCriterion()
```

範例: https://github.com/torch/demos/blob/master/train-a-digit-classifier/train-on-mnist.lua

Torch

- Data format
  - CSV, jons, hdf5, image…
  - 只要可以變成table…
    - Data: 數量 x channel x h x w
    - Labels: 數量
      - Note: Lua index 從1開始
        Label 從1開始
- 過程

```
<mnist> training data:
{
  data : FloatTensor - size: 2000x1x32x32
  normalize : function: 0x41e795f8
  labels : ByteTensor - size: 2000
  normalizeGlobal : function: 0x418f8a98
  size : function: 0x41dfdd60
}
<mnist> loading only 1000 examples
<mnist> done
<mnist> testing data:
{
  data : FloatTensor - size: 1000x1x32x32
  normalize : function: 0x4188c800
  labels : ByteTensor - size: 1000
  normalizeGlobal : function: 0x40b3e6f0
  size : function: 0x405e45e0
}
```

```
ConfusionMatrix:
[[      83       0       0       0       0       0       2       0       0       0]   97.647%   [class: 1]
 [       0     124       1       0       0       0       0       0       1       0]   98.413%   [class: 2]
 [       1       0     107       0       0       0       3       3       2       0]   92.241%   [class: 3]
 [       0       0       3      97       0       4       1       1       1       0]   90.654%   [class: 4]
 [       0       1       0       0     100       1       2       0       0       6]   90.909%   [class: 5]
 [       0       0       0       3       0      83       1       0       0       0]   95.402%   [class: 6]
 [       3       0       0       0       1       0      83       0       0       0]   95.402%   [class: 7]
 [       0       0       5       1       0       0       0      91       0       2]   91.919%   [class: 8]
 [       1       0       1       3       1       1       1       2      78       1]   87.640%   [class: 9]
 [       0       0       1       1       0       0       0       3       2      87]]  92.553%   [class: 10]
 + average row correct: 93.278186917305%
 + average rowUcol correct (VOC measure): 87.339038848877%
 + global correct: 93.3%
```

```
<trainer> online epoch # 3 [batchSize = 10]
[==================== 1161/2000 ..............]  ETA: 17s9ms | Step: 20ms
```

Torch

- With GPU
  - CUDA, cuDNN
  - model:cuda()
  - trainData.data = trainData.data:cuda()
  - trainData.labels = trainData.labels:cuda()
  - testData.data = testData.data:cuda()
  - testData.labels = testData.labels:cuda()
  - criterion = criterion:cuda()

HW5

- Dataset:
  1. 0-9
  2. A-Z
  3. a-z
  4. 符號: ?~/!@#$%^&*()+\
  5. 中文字: 人 價 名 品 單 家 應 數 泉 現 肉 計 量 面 香 件 合 味 商 大 店 收 樂 牛 總 號 豆 金 額
  6. 共106類
  7. Dropbox:

     https://www.dropbox.com/sh/6kqon64xx8g3del/AAB9erdywyc0amWuPzx7DLBIa?dl=0

- 用LeNet 訓練，將dataset分為training和validation
- 12/20以前繳交log檔到104753010@nccu.edu.tw

下週PR為viscovery 參訪