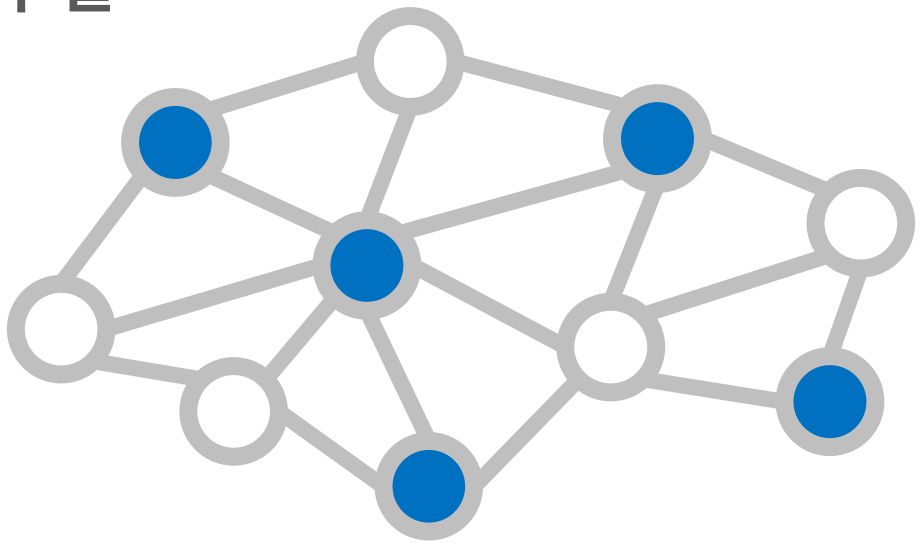
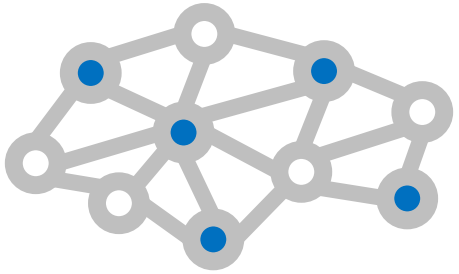
Caffe 실습

서울대학교 융합과학기술대학원
패턴 인식 및 컴퓨터 지능 연구실

박성헌, 황지혜, 유재영



MIPAL aboratory
machine intelligence
&pattern analysis

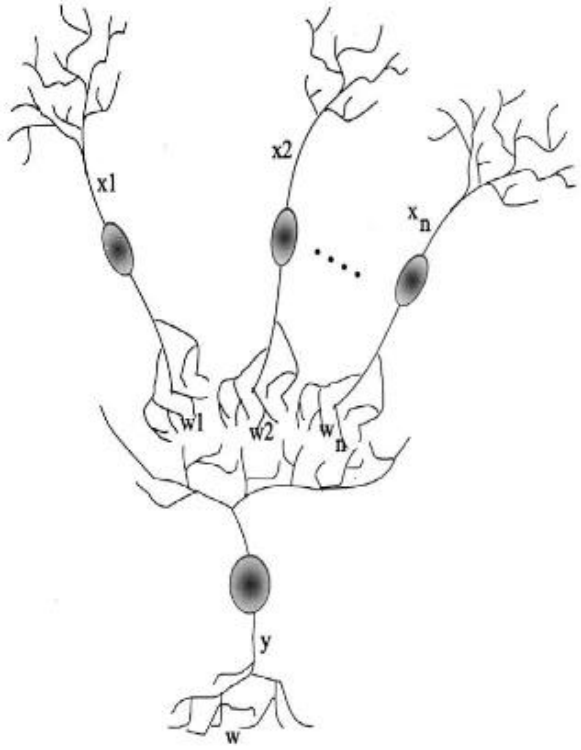


Contents

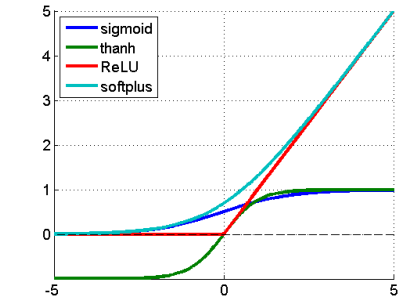
- Caffe 설치
- Caffe 를 이용한 CNN 학습 및 테스트

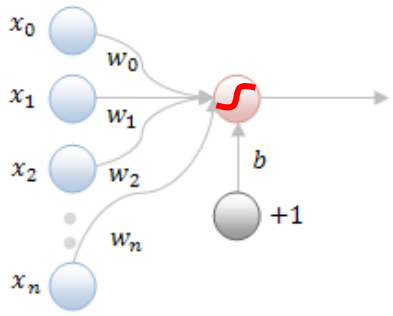
Deep Learning

- Deep Neural Net을 이용한 학습 방법

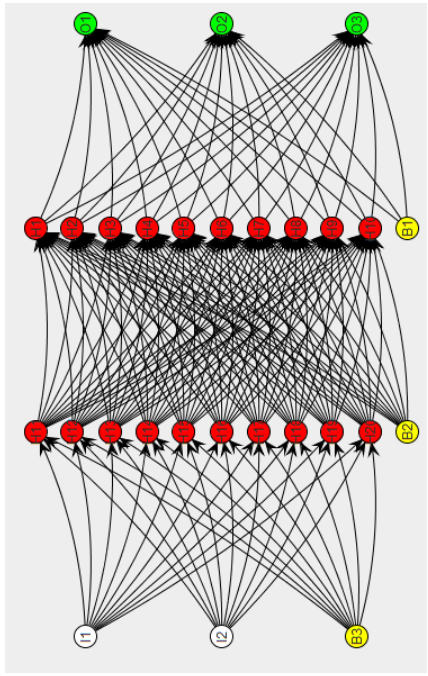


Neuron

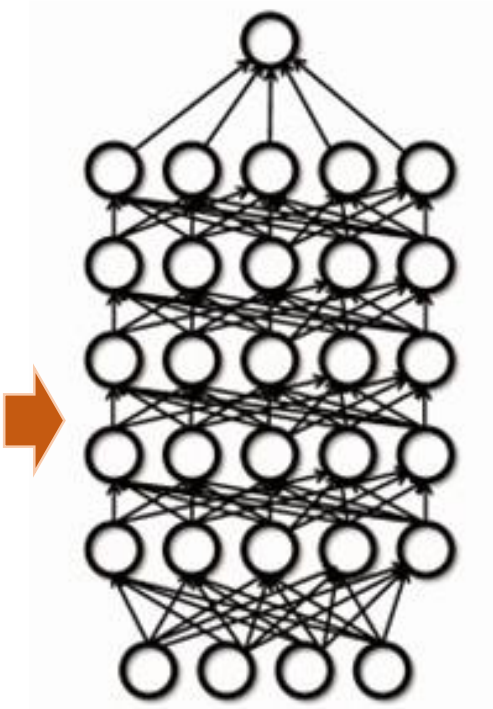




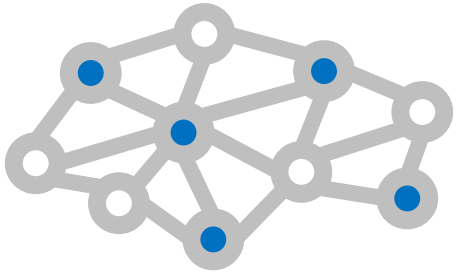
Perceptron



Multi-layer perceptron



Deep neural network

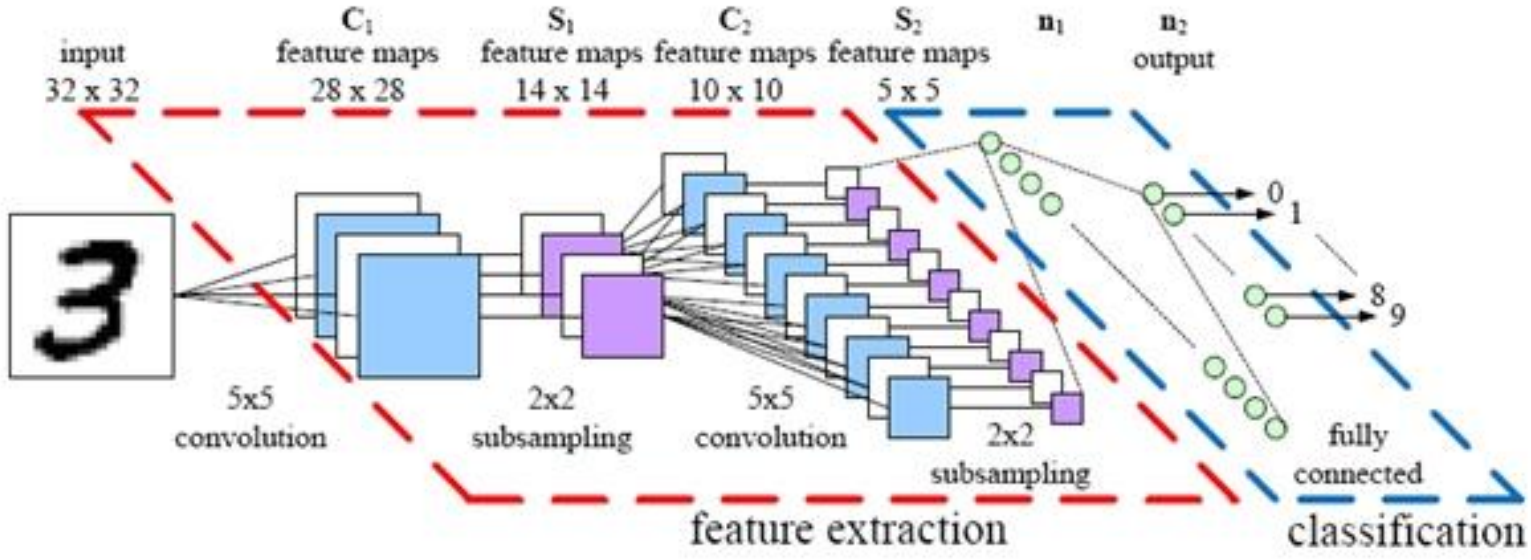


Why Deep Learning?

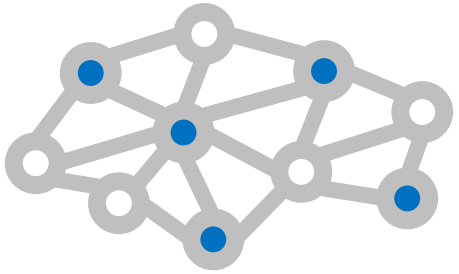
- Performance
  - 영상 인식, 음성 인식 등 다양한 분야에서 최고 성능을 보여줌
- End-to-End learning
  - 데이터와 label만 지정해주면 자동으로 학습이 가능
- Speed
  - GPGPU을 이용하기 좋은 구조
- Versatility
  - 같은 framework를 다양한 분야에 적용 가능

Convolutional Neural Network

- Convolutional Layer 와 Pooling Layer가 핵심 역할
- 영상을 다루기 적합한 Neural Network



http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg



CNN Tutorials

- Stanford CNN Tutorial (Andrew Ng)
  - http://deeplearning.stanford.edu/tutorial/
  - CNN과 Neural Net의 기본 작동원리를 배우고 Matlab를 이용해 직접 구현해 볼 수 있는 tutorial

- Stanford CNN course (Fei-Fei Li & Andrej Karpathy)
  - http://cs231n.Stanford.edu/
  - https://www.youtube.com/playlist?list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC
  - CNN 및 RNN의 기초부터 최신 연구내용까지 다룸

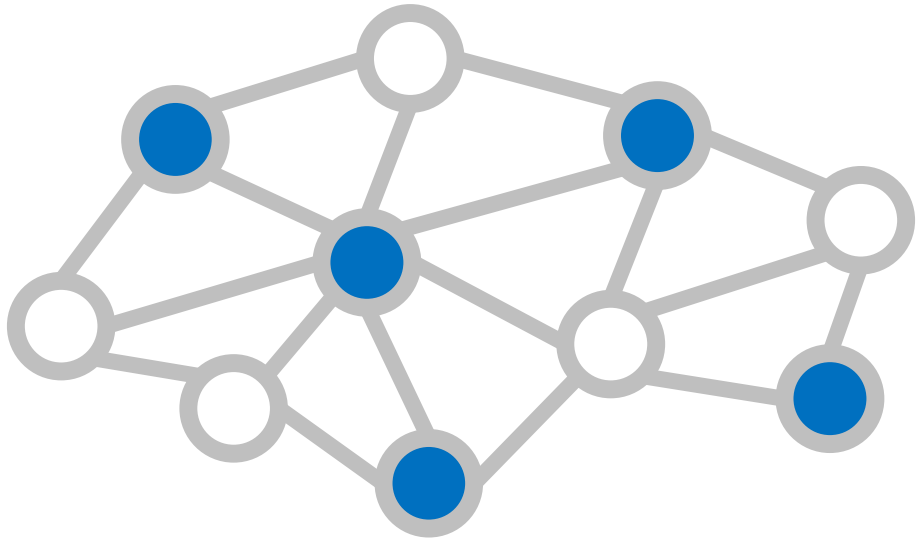
Maximally accurate | Maximally specific

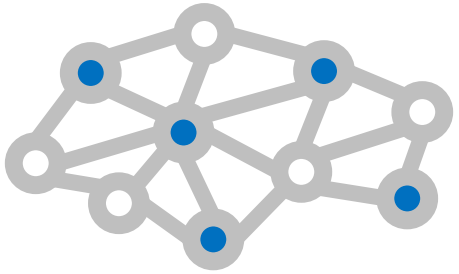
espresso	2.23192
coffee	2.19914
beverage	1.93214
liquid	1.89367
fluid	1.85519

Caffe

Convolutional Architecture for Fast Feature Embedding

MIPAL aboratory
machine intelligence
&pattern analysis





Caffe Overview

- UC Berkeley BVLC (Berkeley Vision and Learning Center)에서 제작
  - 2+ years
  - 1,000+ citations, 150+ contributors

Yangqing Jia

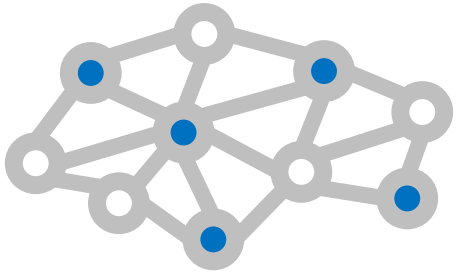
Evan Shelhamer

Travor Darrell

• • •

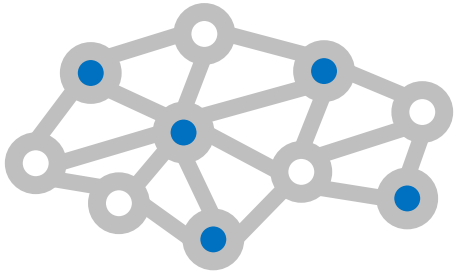
Open-source contributors

Images from BVLC Caffe tutorial



Caffe Overview

- C++, CUDA 로 짜여있음(Matlab, Python wrapper도 존재)
- Open Source Community
  - https://github.com/BVLC/caffe
- BSD License (수정, 재배포, 상업적 사용 등 가능)
- GPGPU acceleration 적용
- Fast, well-tested code
- Many network models available!



Caffe - How Fast?

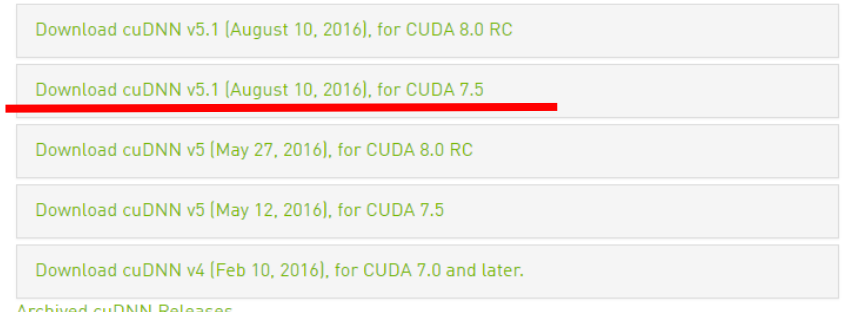
- Speed with Krizhevsky's 2012 model:
  - 2 ms/image on K40 GPU
  - <1 ms inference with Caffe + cuDNN v4 on Titan X
  - 72 million images/day with batched IO
  - 8-core CPU: ~20 ms/image Intel optimization in progress
- 9k lines of C++ code (20k with tests)

Slides from BVLC Caffe tutorial

Caffe Installation (Windows)

- Requirements
  - Visual Studio 2013

- For GPU acceleration
  - CUDA 7.5 (https://developer.nvidia.com/cuda-toolkit)
  - cuDNN v5 (registration 필요) (https://developer.nvidia.com/cudnn)
    - CUDA가 설치된 경로에 파일 추가
      - Ex) C:₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v7.5



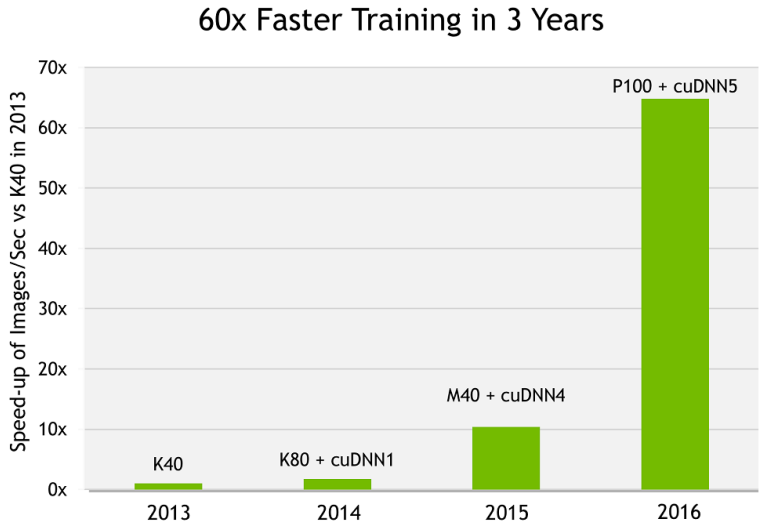
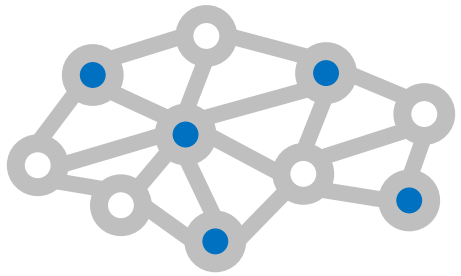
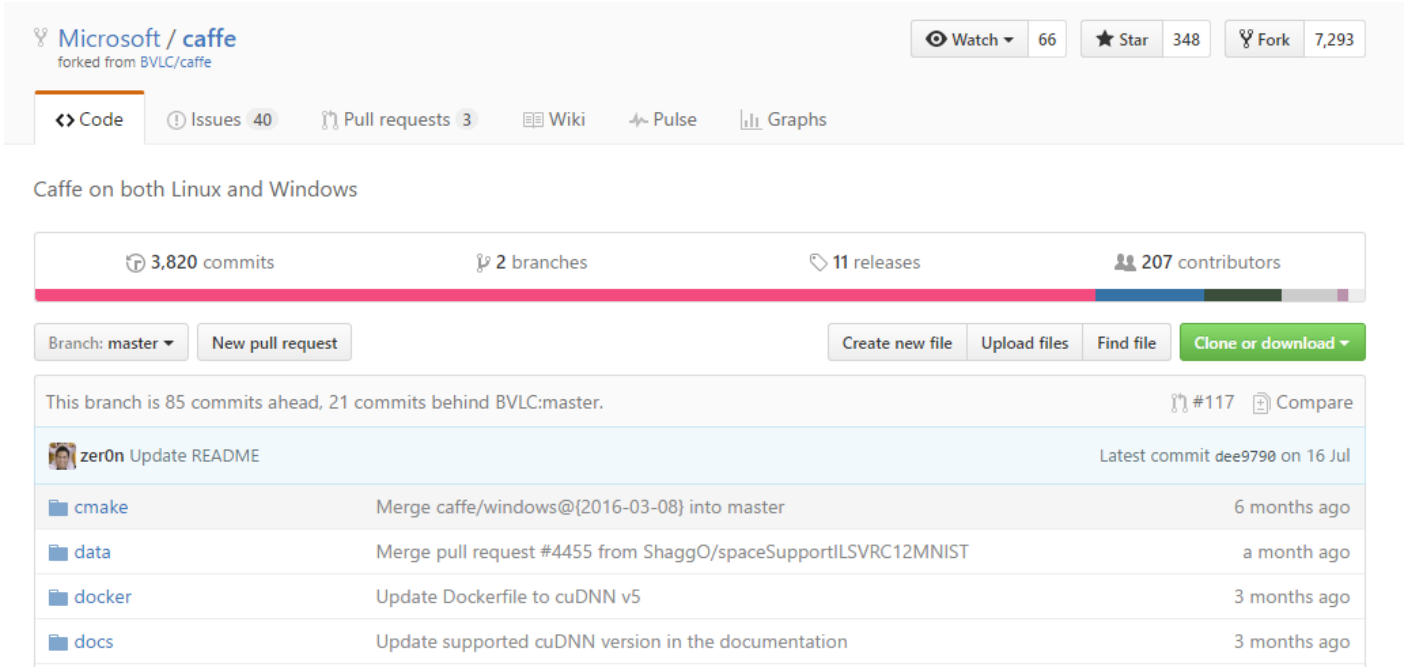


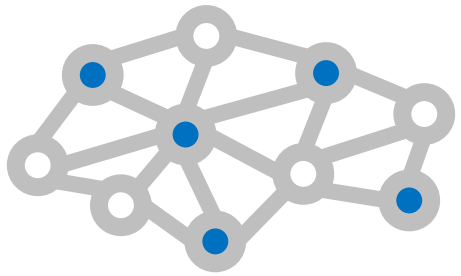
Image from https://developer.nvidia.com/cudnn



Caffe Installation (Windows)

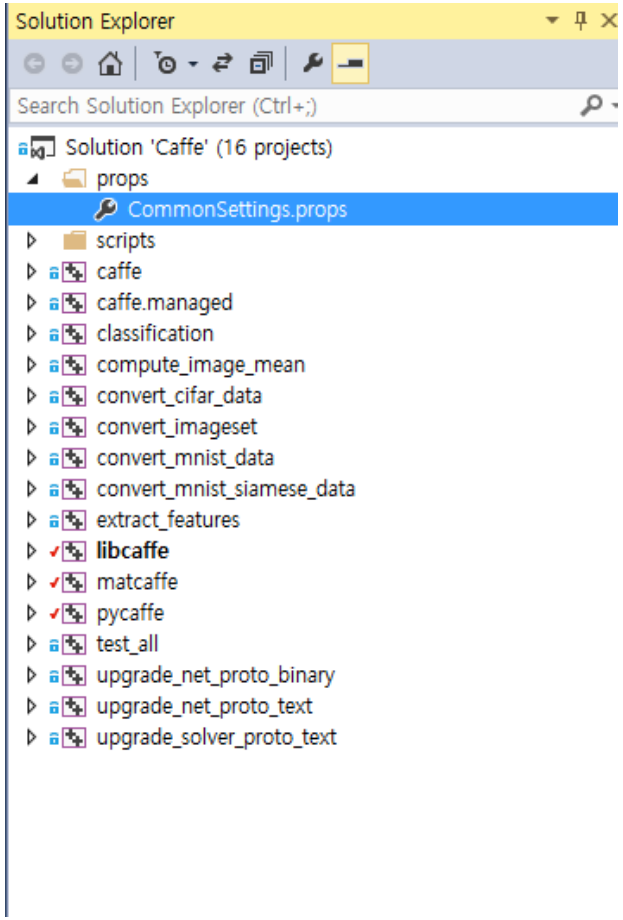
- Caffe Windows branch from Microsoft
  - https://github.com/microsoft/caffe

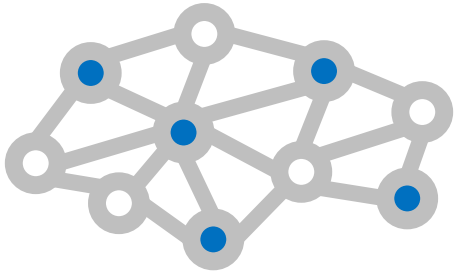




Caffe Installation (Windows)

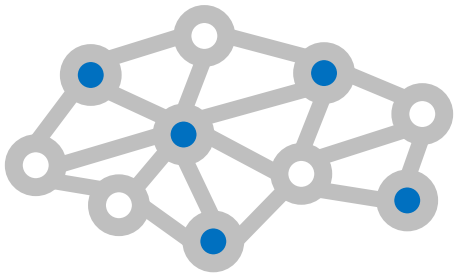
- Setting (CPU build)
  - windows/CommonSettings.props.example
  - CommonSettings.props로 이름 변경
  - Release / x64 로 Build target 설정
  - GPU를 쓰지 않는 경우
    - `<CpuOnlyBuild> true`
    - `<UseCuDNN> false`
  - "C4819: 현재 코드 페이지(949)에서 표시할 수 없는 문자가 파일에 들어 있습니다" Warning이 발생하는 경우
    - `<TreatWarningAsError> true -> false`





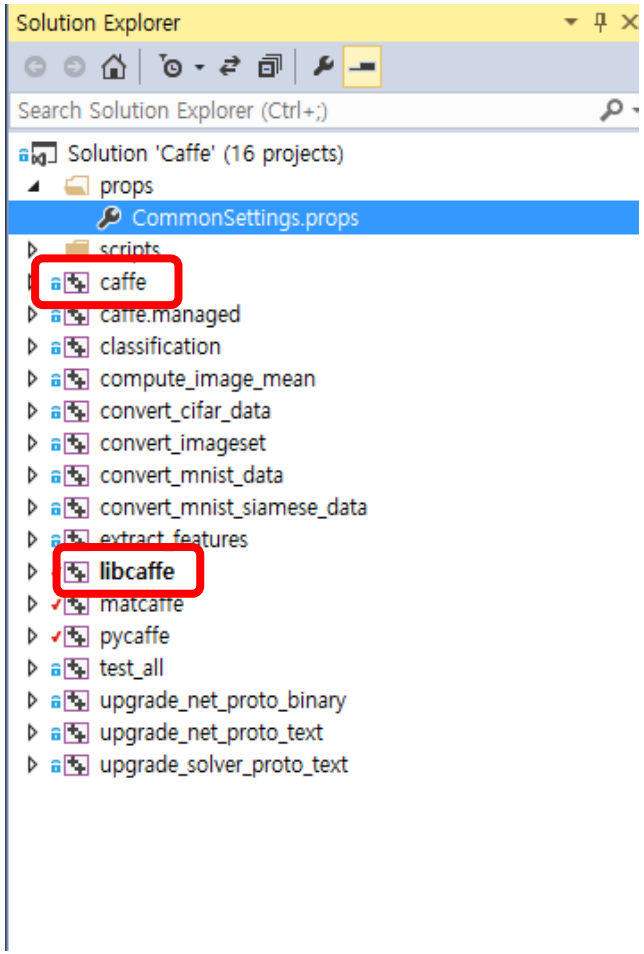
Caffe Installation (Windows)

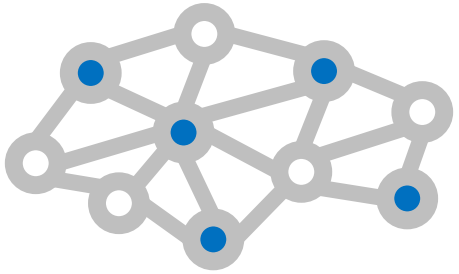
- Setting (GPU build)
  - `<CpuOnlyBuild> false`
  - cuDNN 사용할 경우
    - `<UseCuDNN> true`
  - CUDA version (7.0, 7.5 등)
    - `<CudaVersion>`
  - CUDA compute capability
    - https://developer.nvidia.com/cuda-gpus 에서 확인
    - 장착된 GPU에 맞게 `<CudaArchitecture>` 설정
    - Ex) 3.5인 경우 `compute_35,sm_35`, 5.2인 경우 `compute_52,sm_52`



Caffe Installation (Windows)

- libcaffe 프로젝트 빌드
  - 처음 빌드할 경우 Nuget을 이용해서 필요한 3rd party package들이 다운로드 됨
  - 이후 컴파일 및 빌드에 약 10분 정도 소요
- 완료 후 caffe 프로젝트 빌드





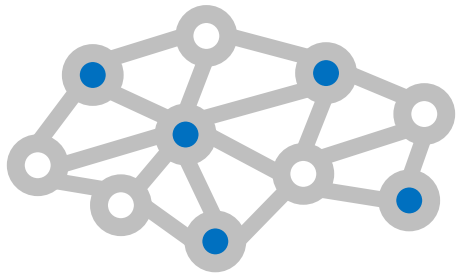
Caffe Installation (Ubuntu)

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

- **General dependencies**
  - sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler
  - sudo apt-get install --no-install-recommends libboost-all-dev

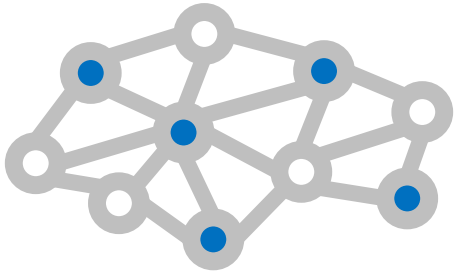
- **BLAS**
  - sudo apt-get install libatlas-base-dev

- **Remaining dependencies, 14.04 / 15.04 / 16.04**
  - sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev



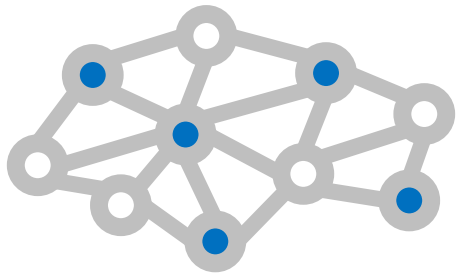
Caffe Installation (Ubuntu)

- **Remaining dependencies, 12.04**
  - glog
    - wget https://google-glog.googlecode.com/files/glog-0.3.3.tar.gz
    - tar zxvf glog-0.3.3.tar.gz
    - cd glog-0.3.3
    - ./configure
    - make && make install
  - gflags
    - wget https://github.com/schuhschuh/gflags/archive/master.zip
    - unzip master.zip
    - cd gflags-master
    - mkdir build && cd build export CXXFLAGS="-fPIC" && cmake .. && make VERBOSE=1
    - make && make install
  - lmdb
    - git clone https://github.com/LMDB/lmdb
    - cd lmdb/libraries/liblmdb
    - make && make install



Caffe Installation (Ubuntu)

- **CUDA 설치 (GPU version only)**
  - CUDA를 지원하는 nvidia GPU 필요: https://developer.nvidia.com/cuda-gpus
  - nvidia graphic driver 설치: http://www.nvidia.com/Download/index.aspx?lang=en-us
  - GPU에 맞는 CUDA version 다운로드: https://developer.nvidia.com/cuda-downloads
    - runfile (local) 다운로드
  - sudo sh cuda_<version>_linux.run --no-opengl-libs
    - Install nvidia driver [Y/N] : N
  - 환경변수 설정
    - open ~/.bashrc
    - 추가 export PATH=/usr/local/cuda-7.5/bin:$PATH
      export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
    - 저장 후 source .bashrc

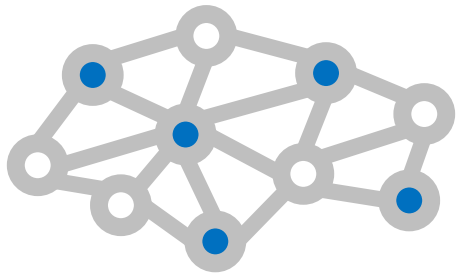


Caffe Installation (Ubuntu)

- **Download Caffe**
  - git clone https://github.com/BVLC/caffe.git

- **Compilation with Make**
  - cp Makefile.config.example Makefile.config # Adjust Makefile.config
    - CPU version: CPU_ONLY := 1
  - make all
    - make all -j8 을 이용해 make 속도를 빠르게 할 수 있음. –j[parallel thread의 수]
  - make test
  - make runtest

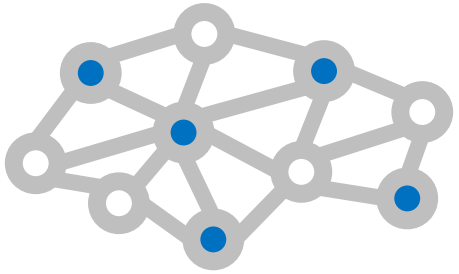
- Ubuntu 15.04 이상 버전에서 **hdf5.h: No such file or directory** 에러 메시지가 발생하면 **Make.config** 파일의 **INCLUDE_DIRS, LIBRARY_DIRS**를 다음과 같이 수정
  - INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include /usr/include/hdf5/serial/
  - LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib /usr/lib/x86_64-linux-gnu/hdf5/serial/



Caffe Installation (Ubuntu)

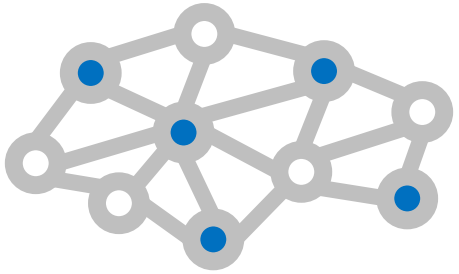
- **Example**
  - Datasets
    - cd $CAFFE_ROOT
    - ./data/mnist/get_mnist.sh
    - ./examples/mnist/create_mnist.sh
  - 실행
    - ./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt
    - CPU 버전일 경우 lenet_solver.prototxt 의 solver mode를 CPU로 변경
  - 결과

```
I0822 11:44:13.930088  8974 sgd_solver.cpp:106] Iteration 9600, lr = 0.00603682
I0822 11:44:19.804344  8974 solver.cpp:228] Iteration 9700, loss = 0.00173592
I0822 11:44:19.804383  8974 solver.cpp:244]     Train net output #0: loss = 0.00173569 (* 1 = 0.00173569 loss)
I0822 11:44:19.804400  8974 sgd_solver.cpp:106] Iteration 9700, lr = 0.00601382
I0822 11:44:25.694238  8974 solver.cpp:228] Iteration 9800, loss = 0.0139525
I0822 11:44:25.694277  8974 solver.cpp:244]     Train net output #0: loss = 0.0139523 (* 1 = 0.0139523 loss)
I0822 11:44:25.694295  8974 sgd_solver.cpp:106] Iteration 9800, lr = 0.00599102
I0822 11:44:31.576386  8974 solver.cpp:228] Iteration 9900, loss = 0.0051976
I0822 11:44:31.576422  8974 solver.cpp:244]     Train net output #0: loss = 0.00519736 (* 1 = 0.00519736 loss)
I0822 11:44:31.576442  8974 sgd_solver.cpp:106] Iteration 9900, lr = 0.00596843
I0822 11:44:37.399930  8974 solver.cpp:454] Snapshotting to binary proto file examples/mnist/lenet_iter_10000.caffemodel
I0822 11:44:37.404800  8974 sgd_solver.cpp:273] Snapshotting solver state to binary proto file examples/mnist/lenet_iter_10000.solverstate
I0822 11:44:37.431008  8974 solver.cpp:317] Iteration 10000, loss = 0.00446007
I0822 11:44:37.431037  8974 solver.cpp:337] Iteration 10000, Testing net (#0)
I0822 11:44:41.099525  8974 solver.cpp:404]     Test net output #0: accuracy = 0.991
I0822 11:44:41.099561  8974 solver.cpp:404]     Test net output #1: loss = 0.0289558 (* 1 = 0.0289558 loss)
I0822 11:44:41.099577  8974 solver.cpp:322] Optimization Done.
I0822 11:44:41.099582  8974 caffe.cpp:254] Optimization Done.
```



Caffe Installation (Mac OSX)

- http://caffe.berkeleyvision.org/install_osx.html 참고



Dataset Link

- 실습에 필요한 데이터셋
  - MNIST (52MB)
    - http://yann.lecun.com/exdb/mnist/
    - Training 및 test 모두 다운로드
  - BVLC alexnet caffemodel (233 MB)
    - http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

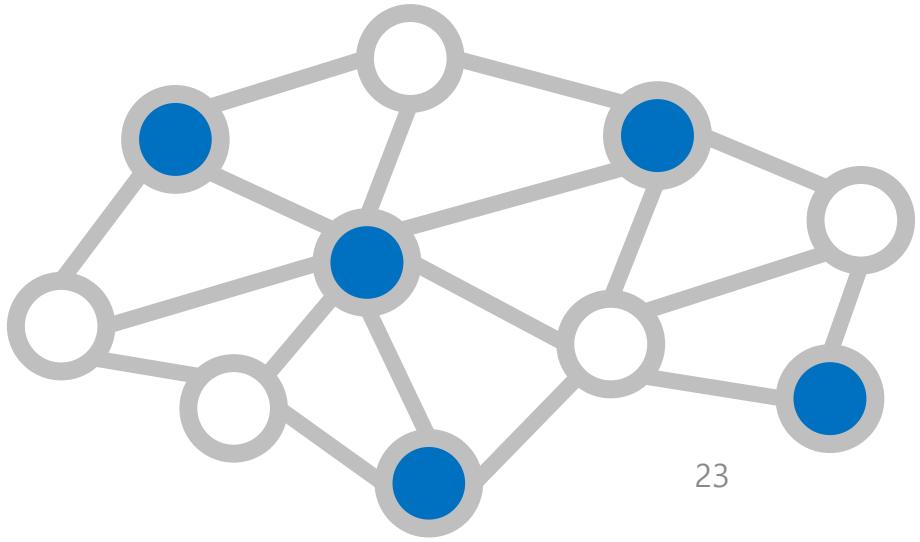
The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

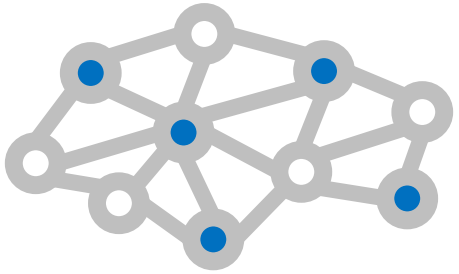
Four files are available on this site:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

Caffe 실습

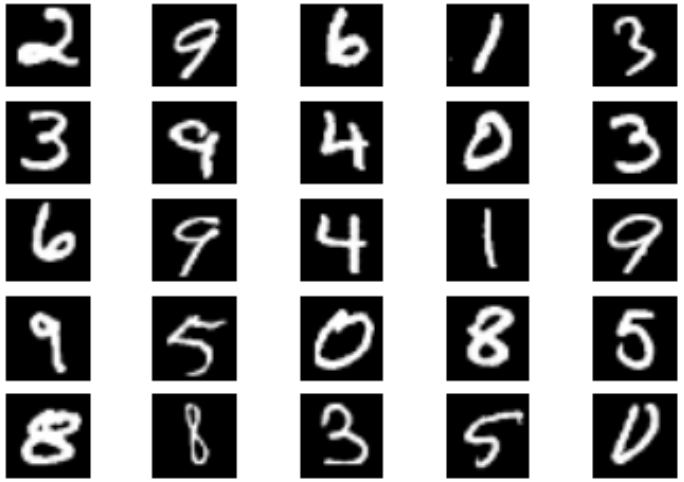


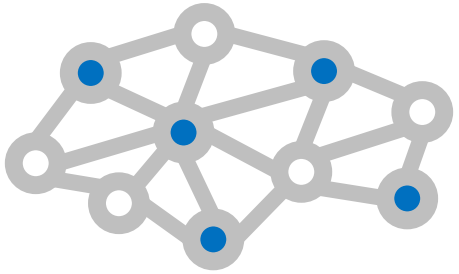
MIPAL aboratory
machine intelligence
&pattern analysis



MNIST Tutorial

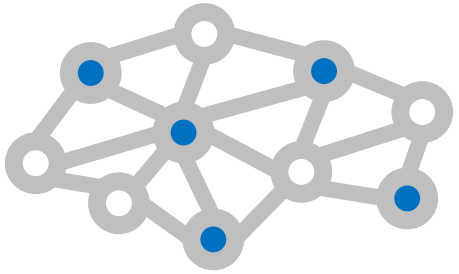
- MNIST dataset [LeCun et al., 1998]
  - 손글씨 인식 데이터셋
  - 32x32 image, 60,000 training samples and 10,000 test samples





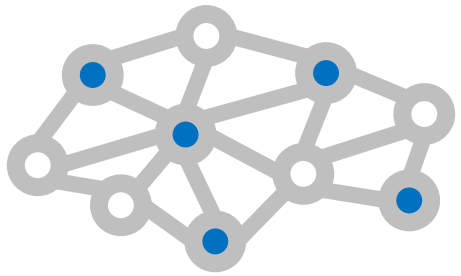
Data preparation

- Caffe에 입력으로 사용되는 데이터 형식
  - LMDB / LEVELDB 를 주로 사용 (빠른 속도)
  - LMDB는 multiple process에서 엑세스 가능, LEVELDB는 불가능
  - Image를 바로 넣거나 HDF5 data를 입력으로 사용할 수도 있음



MNIST Tutorial

- 데이터 준비
  - convert_mnist_data 프로젝트 빌드
  - cmd -> Caffe root directory로 이동
  - Leveldb for training data
    - Build/x64/Release/convert_mnist_data.exe –backend="leveldb" examples/mnist/train-images.idx3-ubyte examples/mnist/train-labels.idx1-ubyte examples/mnist/mnist_train_leveldb
    - Ubuntu(linux)의 경우 Build/x64/Release/convert_mnist_data.exe 대신에 build/examples/mnist/convert_mnist_data.bin 으로 실행
  - Leveldb for test data
    - Build/x64/Release/convert_mnist_data.exe –backend="leveldb" examples/mnist/t10k-images.idx3-ubyte examples/mnist/t10k-labels.idx1-ubyte examples/mnist/mnist_test_leveldb



MNIST Tutorial

- examples/mnist/lenet_solver.prototxt
  - solver_mode : CPU 또는 GPU 설정
- examples/mnist/lenet_train_test.prototxt
  - LMDB -> LEVELDB로 변경
  - 13째 줄

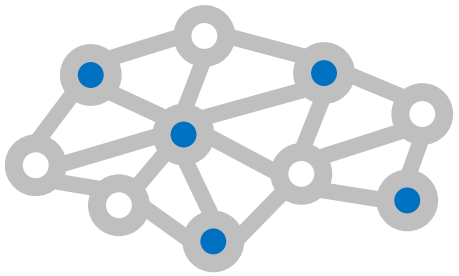
```
data_param {
  source: "examples/mnist/mnist_train_lmdb"
  batch_size: 64
  backend: LMDB
}
```

```
data_param {
  source: "examples/mnist/mnist_train_leveldb"
  batch_size: 64
  backend: LEVELDB
}
```

  - 30째 줄

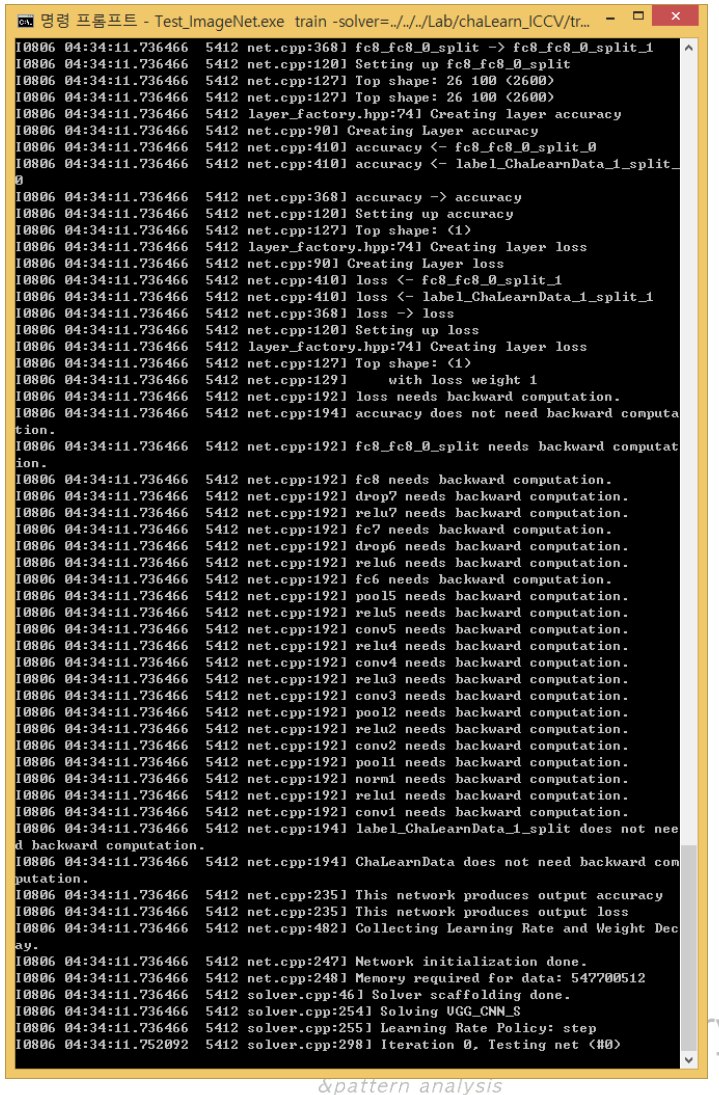
```
data_param {
  source: "examples/mnist/mnist_test_lmdb"
  batch_size: 100
  backend: LMDB
}
```

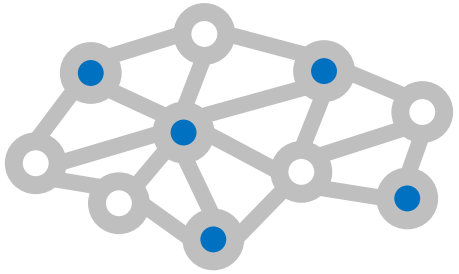
```
data_param {
  source: "examples/mnist/mnist_test_leveldb"
  batch_size: 100
  backend: LEVELDB
}
```



MNIST Tutorial

- LeNet 모델을 이용한 MNIST Training
  - Cmd창 -> Caffe root 폴더로 이동
  - Build/x64/Release/caffe.exe train –solver=examples/mnist/lenet_solver.prototxt 입력
  - Ubuntu(linux)의 경우 Build/x64/Release/caffe.exe 대신에 build/tools/caffe.bin 으로 실행
  - 10,000 iteration 완료시 98~99% accuracy

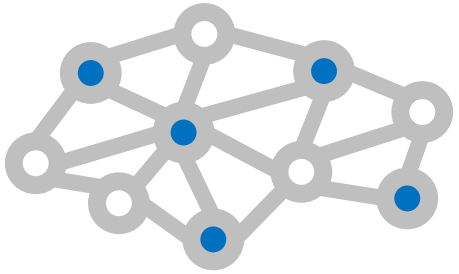




MNIST Tutorial

- Log파일 경로
  - 기본 경로 : C:₩Users₩User_name₩AppData₩Local₩Temp
  - Ubuntu(linux) 기본경로: /tmp
  - log_dir flag로 경로 설정 가능
    - caffe.cpp의 main()함수에서 `FLAGS_log_dir="log_folder";` 추가

```
Iteration 1200, loss = 0.0691786
Train net output #0: loss = 0.0691786 (* 1 = 0.0691786 loss)
Iteration 1200, lr = 0.01
Iteration 1300, loss = 0.134115
Train net output #0: loss = 0.134115 (* 1 = 0.134115 loss)
Iteration 1300, lr = 0.01
Iteration 1400, loss = 0.165894
Train net output #0: loss = 0.165894 (* 1 = 0.165894 loss)
Iteration 1400, lr = 0.01
Iteration 1500, Testing net (#0)
Test net output #0: accuracy = 0.9638
Test net output #1: loss = 0.121973 (* 1 = 0.121973 loss)
Iteration 1500, loss = 0.104802
Train net output #0: loss = 0.104802 (* 1 = 0.104802 loss)
Iteration 1500, lr = 0.01
```

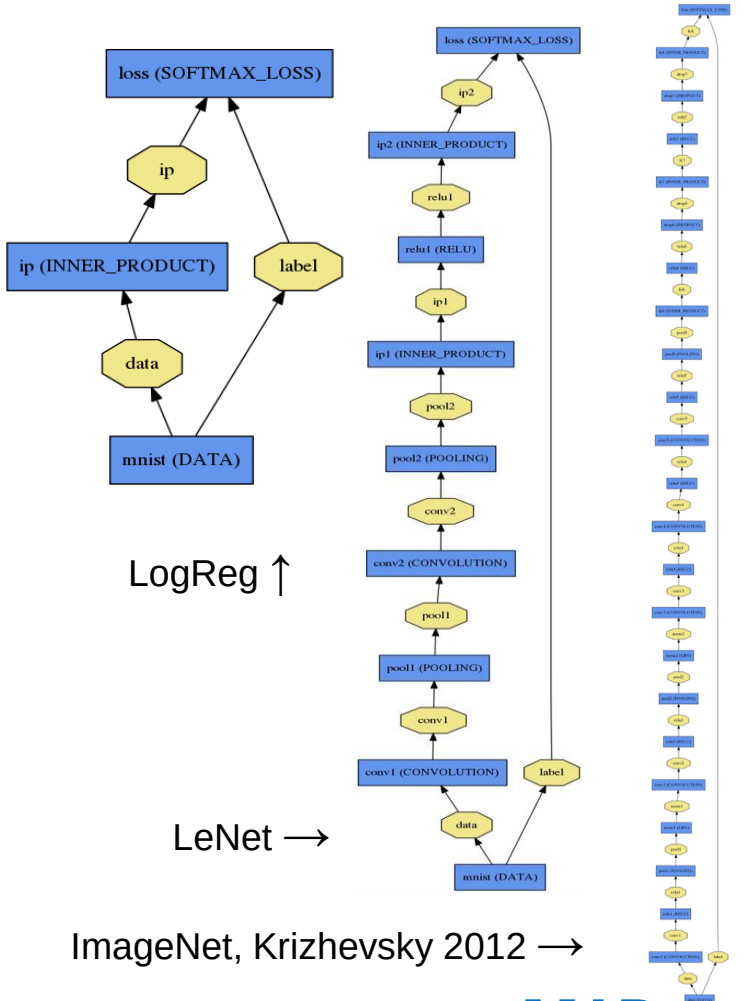


Understanding Caffe Network

- Training/Testing을 위해 보통 두 가지 파일을 정의함
  - Solver 정보를 담은 파일
    - Gradient update를 어떻게 시킬 것인가에 대한 정보를 담음
    - learning rate, weight decay 등의 parameter가 정의됨
    - Test interval, snapshot 횟수 등 정의
  - Network 구조 정보를 담은 파일
    - 실제 CNN 구조 정의
  - 확장자가 .prototxt 파일로 만들어야 함
    - Google Protocol Buffers 기반 (https://developers.google.com/protocol-buffers/)

Understanding Caffe Network

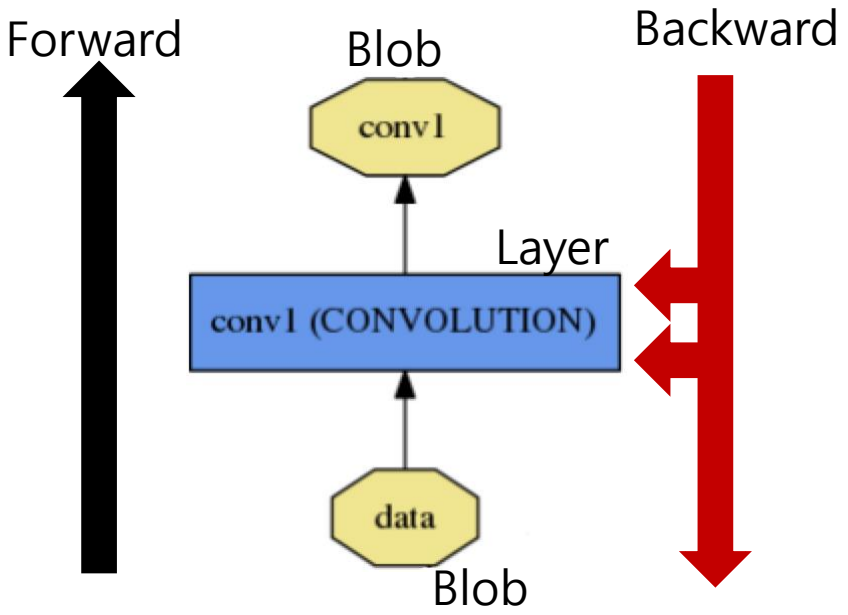
- Net
  - Caffe에서 CNN (혹은 RNN 또는 일반 NN) 네트워크는 'Net'이라는 구조로 정의됨
  - Net은 여러 개의 Layer들이 연결된 구조
  - Directed Acyclic Graph (DAG) 구조만 만족하면 어떤 형태이든 training이 가능함



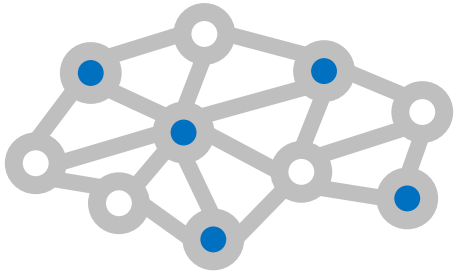
Images from BVLC Caffe tutorial

Understanding Caffe Network

- Layer
  - CNN의 한 '층 ' 을 뜻함
  - Convolution을 하는 Layer, Pooling을 하는 Layer, activation function을 통과하는 layer, input data layer, Loss를 계산하는 layer 등이 있음
  - 소스코드에는 각 layer별로 Forward propagation, Backward propagation 방법이 CPU/GPU 버전별로 구현되어 있음
- Blob
  - Layer를 통과하는 데이터 덩어리
  - Image의 경우 주로 NxCxHxW 의 4차원 데이터가 사용됨 (N : Batch size, C : Channel Size, W : width, H : height)



Images from BVLC Caffe tutorial



Understanding Caffe Network

- Protobuf 파일 들여다보기
  - 예제 : examples/mnist/lenet_train_test.prototxt

Understanding Caffe Network

- 입력 데이터와 관련된 Layer
  - LevelDB data
  - Image data
  - HDF5 data

```
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "examples/mnist/mnist_train_leveldb"
    backend: LEVELDB
    batch_size: 64
  }
  transform_param {
    scale: 0.00390625
    mean_file: mean_mnist.binaryproto
  }
  include: { phase: TRAIN }
}
```

Input Layer는 top이 두개

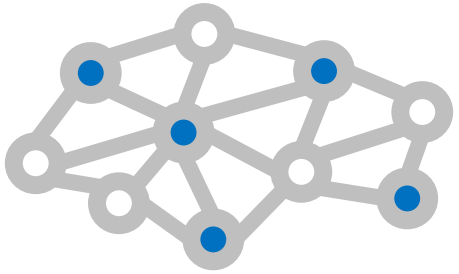
LevelDB 경로

Batch size

1/256

Mean file 빼기

Train과 test시에 쓸 데이터를 따로 지정가능



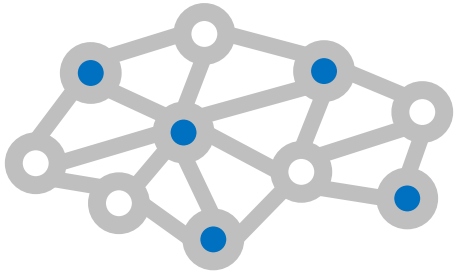
Understanding Caffe Network

- 입력 데이터와 관련된 Layer
  - LevelDB data
  - Image data
    - 이미지를 변환하지 않고 바로 넣을 때 사용
    - LevelDB 또는 LMDB를 이용할 때보다 속도 면에서 약간 느림
  - HDF5 data

Shuffle 여부

Image list정보가 있는 파일
levelDB만들때 입력으로
쓴 파일과 같은 형태

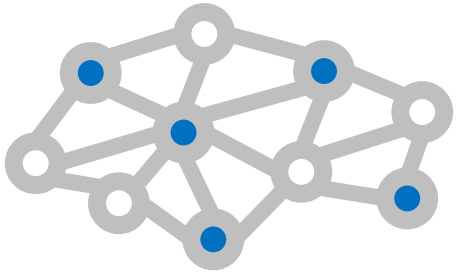
```
layer {
  name: "mnist"
  type: "ImageData"
  top: "data"
  top: "label"
  image_data_param {
    shuffle: true
    source: "examples/mnist/dataList.txt"
    batch_size: 64
  }
  transform_param {
    scale: 0.00390625
    mean_file: mean_mnist.binaryproto
  }
  include: { phase: TRAIN }
}
```



Understanding Caffe Network

- 입력 데이터와 관련된 Layer
  - LevelDB data
  - Image data
  - HDF5 data
    - 영상 이외에 실수 형태의 데이터를 넣을 수 있음

```
layer {
  name: "mnist"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
   source: "examples/mnist/HDF5List.txt"
    batch_size: 64
  }
  transform_param {
    scale: 0.00390625
    mean_file: mean_mnist.binaryproto
  }
  include: { phase: TRAIN }
}
```



Understanding Caffe Network

- Convolution Layer
  - Input size (i1xi2xi3)
  - Output size (o1xo2xo3)
  - Filter size (f1xf2)
  - 일 경우
  - 학습할 parameter 수:
  - o3xi3xf1xf2

o1 = (i1 + 2 x pad_size – f1) / stride + 1

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {lr_mult: 1}
  param {lr_mult: 2}
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    pad: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

Layer별로 Learning rate를 다르게 조정가능. Solver에서 정한 learning rate에 곱해진 값이 해당 layer의 learning rate가 됨
첫번째는 weight에 대한 learning rate,
두번째는 bias에 대한 learning rate

Convolution후 output으로 나오는 feature map 개수

Convolution에 쓰이는 filter의 크기

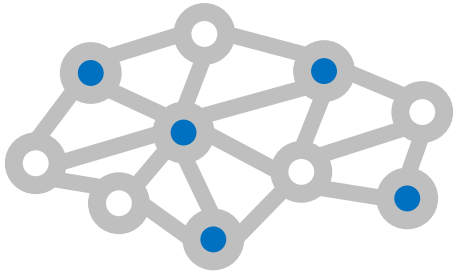
Stride 설정

Padding 설정

Weight에 대한 initialization. Gaussian도 많이 쓰임

Bias에 대한 initialization.
Constant의 경우 value를 함께 지정 가능.
Default 0

MIPAL aboratory
machine intelligence &pattern analysis

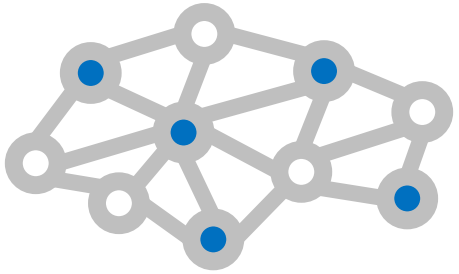


Understanding Caffe Network

- Pooling Layer
  - Size 계산은 convolution 경우와 같음

Max, mean, stochastic 가능

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

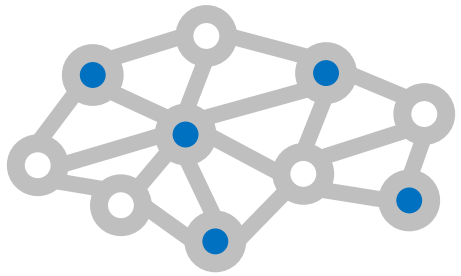


Understanding Caffe Network

- Activation Layer

RELU, sigmoid, tanH 등 가능
RELU는 negative_slope 설정 가능

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



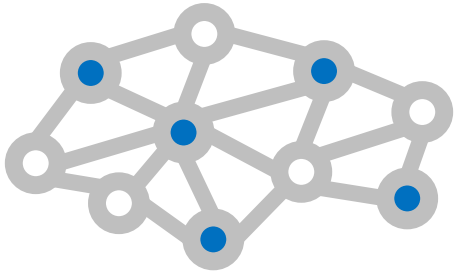
Understanding Caffe Network

- Fully connected layer
  - 일반적인 neural network에서처럼 아래 blob과 위 blob의 모든 뉴런간에 연결이 되어 있는 layer

Fully connected layer output 뉴런 개수 →

Initialization from Gaussian distribution →

```
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: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
    }
  }
}
```



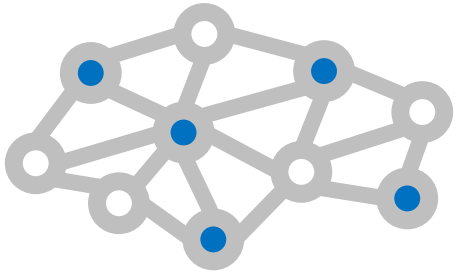
Understanding Caffe Network

- Loss layer
  - 가장 마지막 layer로 label과 비교해서 loss를 계산함
  - Cross entropy, Euclidean distance 등 다양한 loss가 정의되어 있음

Classification 문제에는 Softmax를 주로 사용 (Cross Entropy Loss)

Loss Layer는 bottom이 두개

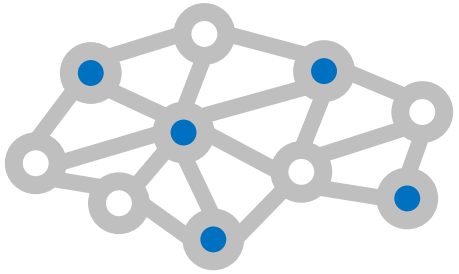
```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```



Understanding Caffe Network

- Accuracy layer
  - Test 시에 Accuracy를 표시하기 위해 주로 사용

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

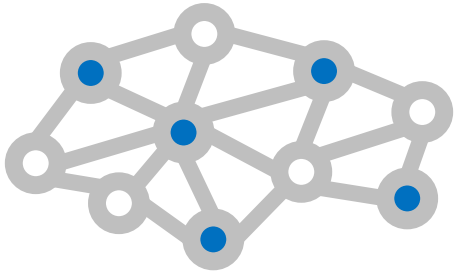


Understanding Caffe Network

- Dropout Layer
  - [Hinton et al. NIPS 2012] 논문에 소개된 내용으로 over-fitting을 방지하고 generalization 효과가 좋음
  - 주로 Fully connected layer에 사용

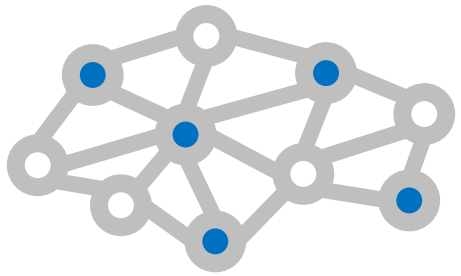
보통 0.5 사용 →

```
layer {
  name: "drop7"
  type: "Dropout"
  bottom: "fc7-conv"
  top: "fc7-conv"
  dropout_param {
    dropout_ratio: 0.5
  }
}
```



Understanding Caffe Network

- 많은 새로운 Layer와 Option이 추가되고 있음
  - http://caffe.berkeleyvision.org/tutorial/layers.html 에서 많이 사용되는 Layer와 그 사용법을 볼 수 있음
  - 현재 50개 이상의 Layer 종류가 존재함
  - 최근 추가된 Layer는 caffe.proto 파일 및 github의 discussion 등을 통해 알 수 있음



Understanding Caffe Solver

- Solver 정의하기

```
net: "examples/mnist/lenet_train_test.prototxt"
test_iter: 100
test_interval: 500
type: "SGD"
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"
gamma: 0.0001
power: 0.75
display: 100
max_iter: 10000
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
solver_mode: GPU
```

Net 구조를 정의한 prototxt 파일

Test시에 iteration 횟수. Test_iter x batch_size만큼 test를 함

몇번 iteration돌때마다 test를 할 것인가?

Solver type

Learning rate

momentum

Weight decay

Learning rate 변화를 어떻게 시킬 것인가

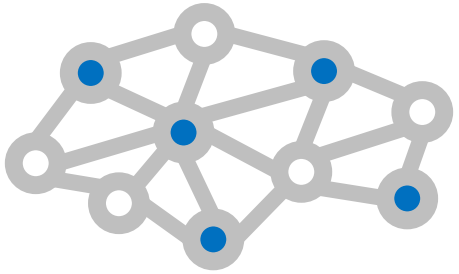
Loss를 보여주는 iteration 횟수

총 training iteration 수

Iteration 횟수마다 기록을 남김.
.caffemodel과 .solverstate파일이 생성됨

Snapshot 파일앞에 붙일 이름

CPU or GPU



Understanding Caffe Solver

- Stochastic gradient descent solver (type: "SGD")
  - 몇 가지 solver가 더 있으나 SGD가 가장 많이 사용됨

$$V_{t+1} = \mu V_t - \alpha \nabla L(W_t)$$

momentum　Learning rate　계산된 gradient

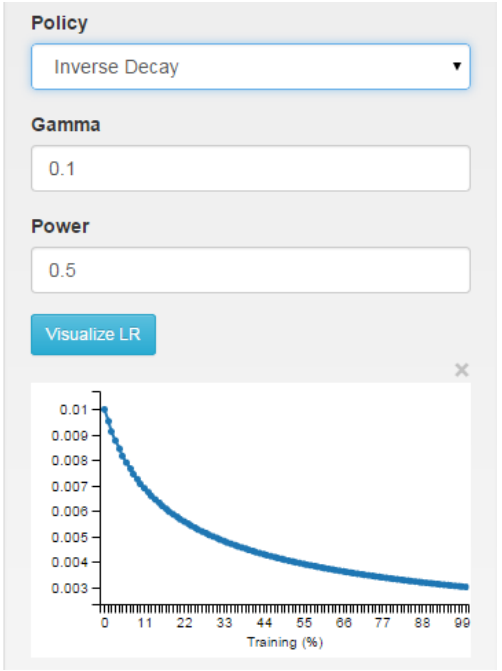
  - 최종 weight update

$$W_{t+1} = W_t + V_{t+1}$$

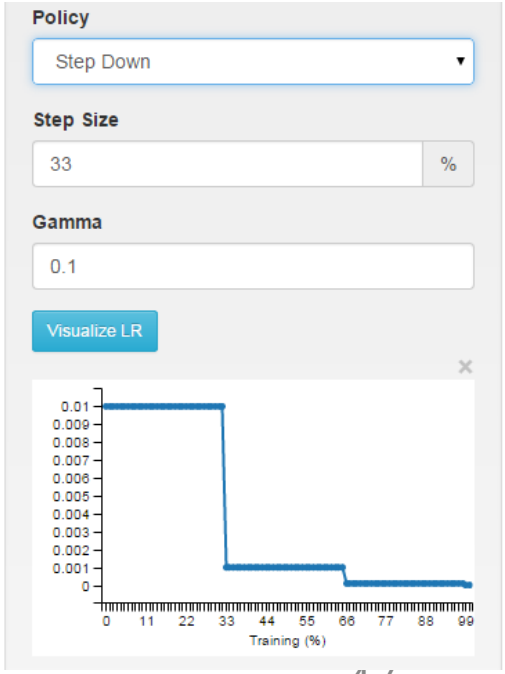
Understanding Caffe Solver

- Learning Rate 결정하기
  - 주로 step이 많이 사용됨.
  - DIGITS 라는 tool을 이용하면 visualization 가능
  - https://github.com/NVIDIA/DIGITS

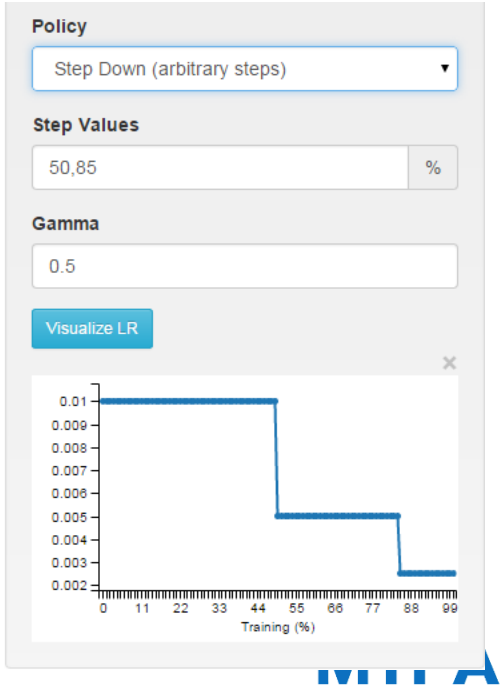
inv



step



multistep

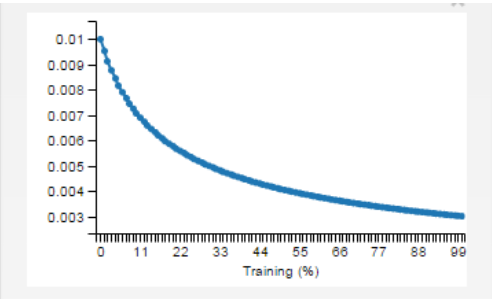


Understanding Caffe Solver

- Learning Rate 결정하기

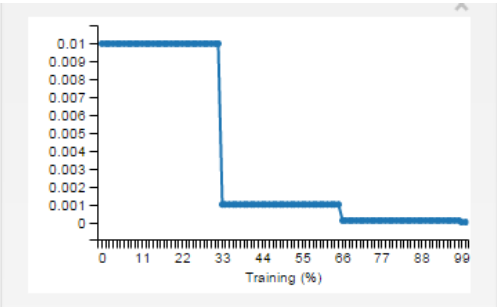
lr_policy: "inv"
gamma: 0.1
power: 0.5
base_lr: 0.01

$\alpha = base\_lr \times (1 + \gamma \times iter)\text{^}(-power)$

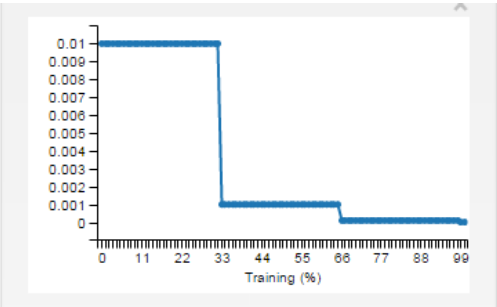


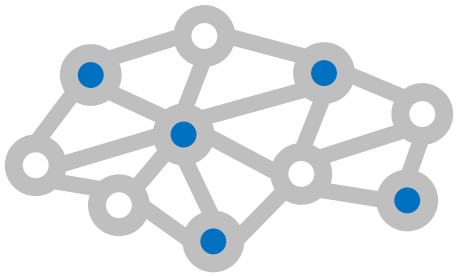
lr_policy: "step"
gamma: 0.1
step: 10000
base_lr: 0.01

$\alpha = base\_lr \times (\gamma\text{^}\lfloor iter/step \rfloor)$



lr_policy: "multistep"
gamma: 0.1
stepvalue: 5000
stepvalue: 8000
base_lr: 0.01





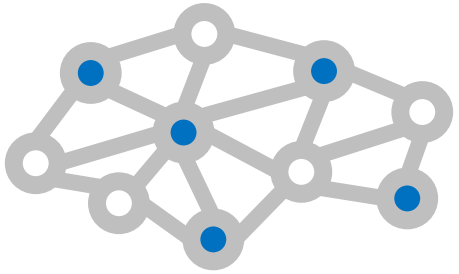
Understanding Caffe Solver

- RMS Prop (type: "RMSProp")
  - Gradient가 oscillate할 경우 $(1-\delta)$만큼 곱함, 그렇지 않을 경우 δ만큼 더해줌

$$(v_t)_i = \begin{cases} (v_{t-1})_i + \delta, & \text{if } \nabla L(W_t)_i \nabla L(W_{t-1})_i > 0 \\ (v_{t-1})_i \times (1-\delta) & \text{otherwise} \end{cases}$$

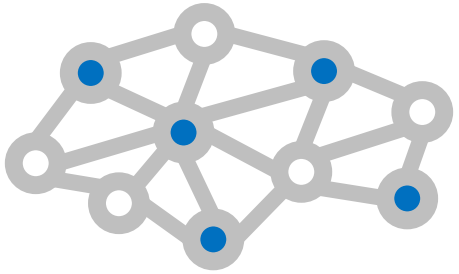
  - 최종 weight update

$$(W_{t+1})_i = (W_t)_i - \alpha (v_t)_i$$



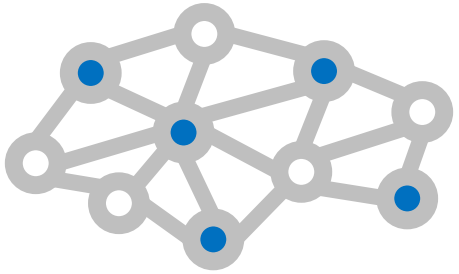
Understanding Caffe Solver

- Rule of thumb
  - Momentum = 0.9
  - Weight decay = 0.0005
  - Base learning rate = 0.01
- 그 외 다양한 Solver의 Optimization algorithm은 http://caffe.berkeleyvision.org/tutorial/solver.html 에서 확인 가능



Terminal Interface

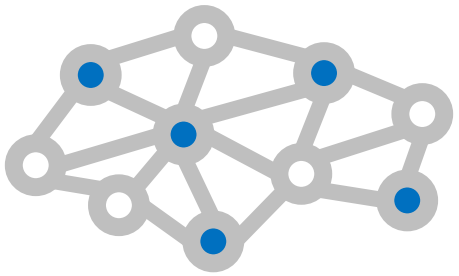
- Training
  - caffe.exe train –solver=solver_file.prototxt (Ubuntu: caffe.bin)
- Testing
  - Backward propagation없이 forward propagation을 통한 결과값만 출력
  - caffe.exe test –gpu=0 –iterations=100 –weights=weight_file.caffemodel –model=net_model.prototxt
  - -model은 solver가 아닌 net파일을 입력으로 줘야 함
  - -weights는 미리 학습된 weight 파일 (.caffemodel 확장자)
  - -iterations 옵션만큼 iteration 수행
- caffe.exe –help 옵션으로 flag에 대한 도움말을 볼 수 있음



MNIST Tutorial

- 로그 들여다보기

```
I0823 14:33:56.829655  2040 net.cpp:408] mnist -> data
I0823 14:33:56.832659  2040 net.cpp:408] mnist -> label
I0823 14:33:56.842664 15152 common.cpp:36] System entropy source not available, using fallback algorithm to generate seed instead.
I0823 14:33:56.875715 15152 db_leveldb.cpp:18] Opened leveldb examples/mnist/mnist_train_leveldb
I0823 14:33:56.962749  2040 data_layer.cpp:41] output data size: 64,1,28,28
I0823 14:33:56.967756  2040 net.cpp:150] Setting up mnist
I0823 14:33:56.968763  2040 net.cpp:157] Top shape: 64 1 28 28 (50176)
I0823 14:33:56.971756  2040 net.cpp:157] Top shape: 64 (64)
I0823 14:33:56.974786  2040 net.cpp:165] Memory required for data: 200960
I0823 14:33:56.978761  2040 layer_factory.hpp:77] Creating layer conv1
I0823 14:33:56.982764  2040 net.cpp:100] Creating Layer conv1
I0823 14:33:56.983767 12856 common.cpp:36] System entropy source not available, using fallback algorithm to generate seed instead.
I0823 14:33:56.983767  2040 net.cpp:434] conv1 <- data
I0823 14:33:56.986766  2040 net.cpp:408] conv1 -> conv1
I0823 14:33:56.990768  2040 net.cpp:150] Setting up conv1
I0823 14:33:56.990768  2040 net.cpp:157] Top shape: 64 20 24 24 (737280)
I0823 14:33:56.991770  2040 net.cpp:165] Memory required for data: 3150080
I0823 14:33:56.992770  2040 layer_factory.hpp:77] Creating layer pool1
I0823 14:33:56.993770  2040 net.cpp:100] Creating Layer pool1
I0823 14:33:56.994771  2040 net.cpp:434] pool1 <- conv1
I0823 14:33:56.994771  2040 net.cpp:408] pool1 -> pool1
I0823 14:33:56.996780  2040 net.cpp:150] Setting up pool1
I0823 14:33:56.997779  2040 net.cpp:157] Top shape: 64 20 12 12 (184320)
I0823 14:33:56.998778  2040 net.cpp:165] Memory required for data: 3887360
```



Memory Management

- CUDA out of memory error (code 2)
  - CNN구조를 바꾸거나 Batch size를 줄여줘야 한다.
  - C:₩Program Files₩NVIDIA Corporation₩NVSMI₩nvidia-smi.exe에서 현재 메모리 사용현황 확인 가능

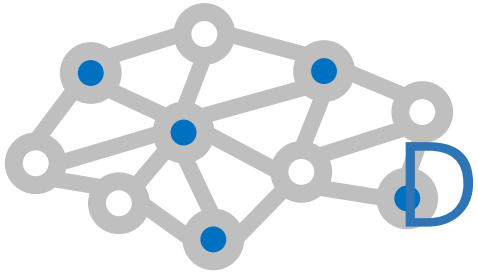
```
I0807 22:14:15.959929  3352 net.cpp:247] Network initialization done.
I0807 22:14:15.959929  3352 net.cpp:248] Memory required for data: 5392743432
I0807 22:14:15.959929  3352 solver.cpp:46] Solver scaffolding done.
I0807 22:14:15.959929  3352 solver.cpp:254] Solving VGG_CNN_S
I0807 22:14:15.960930  3352 solver.cpp:255] Learning Rate Policy: step
I0807 22:14:16.000941  3352 solver.cpp:298] Iteration 0, Testing net (#0)
I0807 22:15:13.532956  3352 solver.cpp:347]     Test net output #0: accuracy = 0

I0807 22:15:13.532956  3352 solver.cpp:347]     Test net output #1: loss = 4.605
17 (* 1 = 4.60517 loss)
F0807 22:15:15.642501  3352 syncedmem.cpp:51] Check failed: error == cudaSuccess
 (2 vs. 0)  out of memory
*** Check failure stack trace: ***

D:₩Caffe_executable_150624₩caffe-windows₩bin>
```

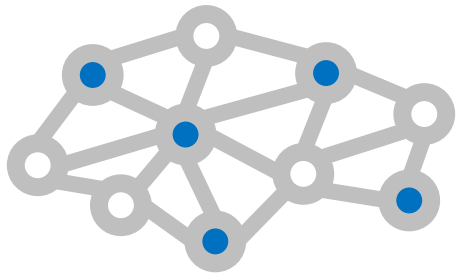
```
c:₩Program Files₩NVIDIA Corporation₩NVSMI>nvidia-smi.exe
Fri Aug 07 22:22:01 2015
+------------------------------------------------------+
| NVIDIA-SMI 353.62     Driver Version: 353.62         |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX TIT... WDDM  | 0000:01:00.0      On |                  N/A |
| 47%   83C    P2   181W / 250W |  11229MiB / 12288MiB |     99%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0         4  C+G   Insufficient Permissions                       N/A      |
|    0       940  C+G   Insufficient Permissions                       N/A      |
|    0      1252    C   ...50624₩caffe-windows₩bin₩Test_ImageNet.exe N/A      |
|    0      2280  C+G   ...les₩Microsoft Office₩Office15₩OUTLOOK.EXE N/A      |
|    0      3224  C+G   ...Visual Studio 12.0₩Common7₩IDE₩devenv.exe N/A      |
|    0      3920  C+G   C:₩Windows₩Explorer.EXE                       N/A      |
|    0      5496  C+G   ...x86)₩Google₩Chrome₩Application₩chrome.exe N/A      |
|    0      6320  C+G   ...es₩Microsoft Office₩Office15₩POWERPNT.EXE N/A      |
+-----------------------------------------------------------------------------+
```



DIY! Creating Your Own Network

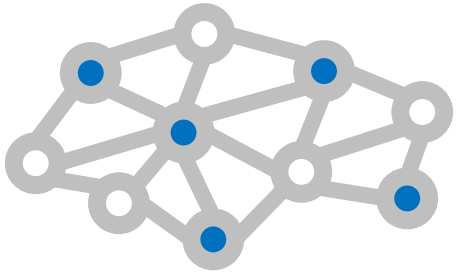
- Convolution layer의 filter 개수 및 kernel size 조절
- Convolution layer 및 Pooling layer 쌓기
- 다양한 Solver를 이용한 Optimization



Data Preparation

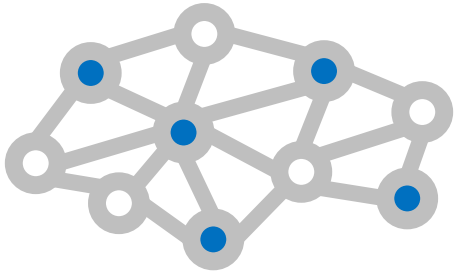
- Dataset 준비하기 (Convert_imageset 프로젝트 빌드)
  - 영상 데이터와 ground truth label을 준비
  - Label은 다음과 같은 형태로 텍스트 파일로 만듦
    - Subfolder1/file1.JPEG 7
    - Subfolder2/file2.JPEG 3
    - Subfolder3/file3.JPEG 4

  - Label은 0부터 시작
  - Shuffle, resize 등의 옵션을 활용
  - 사용법: 실행파일.exe [FLAGS] ROOTFOLDER/ LISTFILE DB_NAME
  - 예시: convert_imageset.exe –backend="leveldb" –shuffle=true imageData/ imageList.txt imageData_levelDB
  - Ubuntu(linux)의 경우 convert_imageset.bin, 옵션은 동일



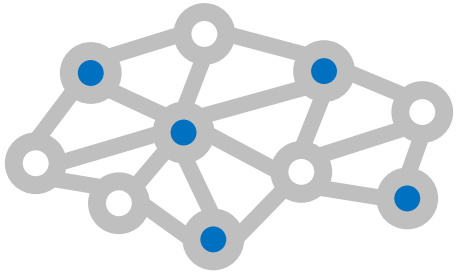
Data Preparation

- Mean image 구하기 (Compute_image_mean 프로젝트 빌드)
  - 대부분의 경우 training, testing 시에 image data에서 mean image를 뺀다
  - LevelDB 또는 LMDB를 이용해서 만듦
  - 사용법: 실행파일.exe [FLAGS] INPUT_DB [OUTPUT_FILE]
  - 예시: compute_image_mean.exe –backend="leveldb" imageData_levelDB mean_imageData.binaryproto
  - Ubuntu(linux)의 경우 compute_image_mean.bin, 옵션은 동일
  - 실행결과 binaryproto 파일이 생성됨.



Terminal Interface

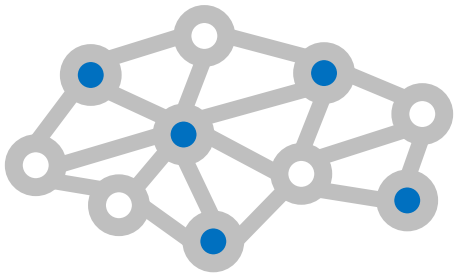
- Training을 중간에 멈춘 뒤 이어서 하고 싶을때
  - Snapshot으로 남겨둔 solverstate파일을 이용 (-snapshot 옵션)
  - caffe.exe train –solver=solver.prototxt -snapshot=lenet_iter_5000.solverstate
- Fine tuning / Transfer learning
  - Pre-trained model을 이용하는 방법
  - Snapshot으로 남겨둔 caffemodel파일을 이용 (-weights 옵션)
  - caffe.exe train –solver=solver.prototxt –weights=lenet_iter_5000.caffemodel
  - Layer 이름을 비교해서 이름이 같은 Layer는 caffemodel파일에서 미리 training된 weight를 가져오고 새로운 layer는 새로 initialization을 해서 학습함.



ImageNet Tutorial

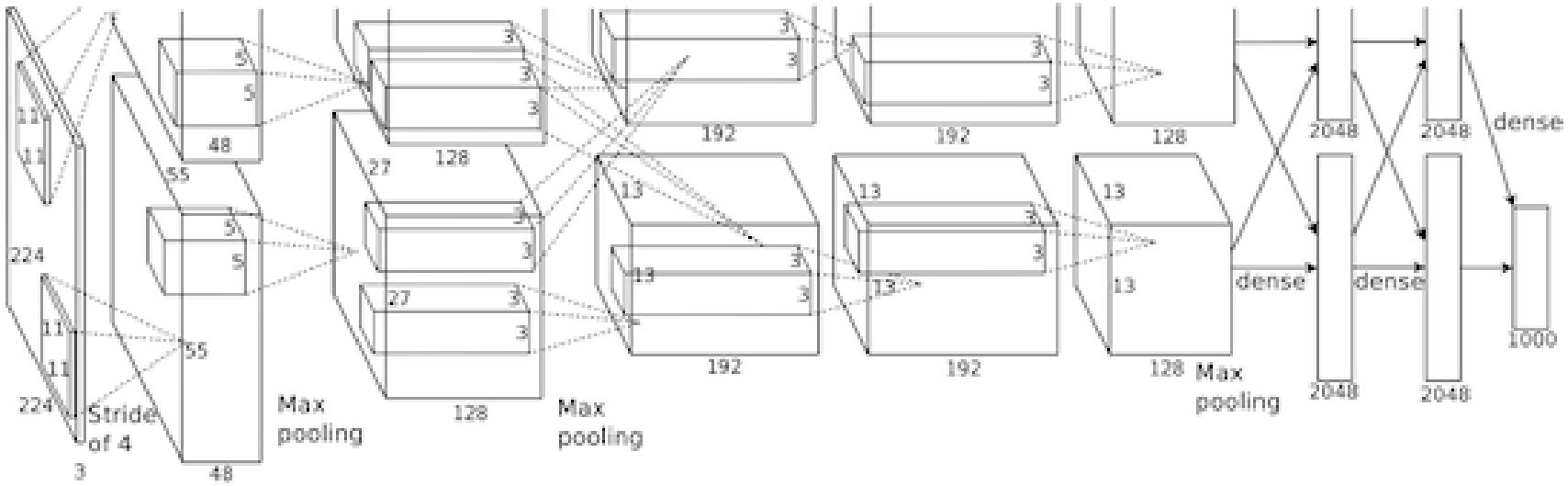
- ILSVRC2012(ImageNet) data set
  - 약 128만 장의 라벨링된 training set, 5만장의 validation set, 10만장의 test set으로 구성
  - 1000개의 class로 구성.

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

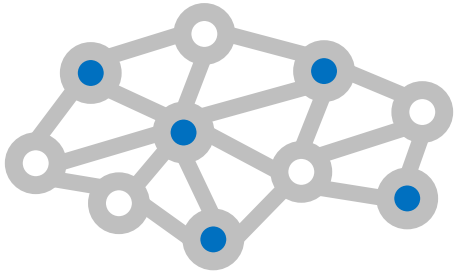


ImageNet Tutorial

- AlexNet
  - 2012년 ImageNet에서 우승한 CNN 모델
  - 40.7% Top 1 Error, 18.2% Top 5 Error on validation set
  - 5개의 convolution network, 3개의 pooling layer, 2개의 fully connected layer로 구성



A Krizhevsky et al., NIPS 2012



ImageNet Tutorial

- Image classification using AlexNet
  - deploy.prototxt : network model 파일. 임의의 입력을 다룰 때 사용.

```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
}
```

  - alexnet.caffemodel : 미리 학습 된 Alexnet 학습 모델.
  - mean.binaryproto: imagenet dataset mean file.
  - label.txt: class label 정보를 담고 있는 txt 파일
  - test.jpg: test 할 이미지 파일.

ImageNet Tutorial

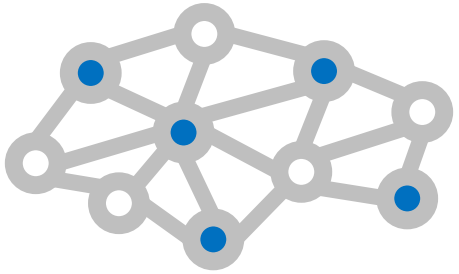
- Image classification using AlexNet
  - classification 프로젝트 빌드
  - Build/x64/Release/classification.exe models/bvlc_alexnet/deploy.prototxt models/bvlc_alexnet/bvlc_alexnet.caffemodel data/ilsvrc12/imagenet_mean.binaryproto data/ilsvrc12/synset_words.txt data/ilsvrc12/test_image.jpg
  - Ubuntu(linux): Build/x64/Release/classification.exe 를 build/examples/cpp_classification/classification.bin 으로 대체

최종 top5 예측 결과

```
---------- Prediction for data₩ilsvrc12₩test_image.jpg ----------
0.6162 - "n02123045 tabby, tabby cat"
0.2667 - "n02124075 Egyptian cat"
0.1028 - "n02123159 tiger cat"
0.0141 - "n02127052 lynx, catamount"
0.0000 - "n02120505 grey fox, gray fox, Urocyon cinereoargenteus"
```

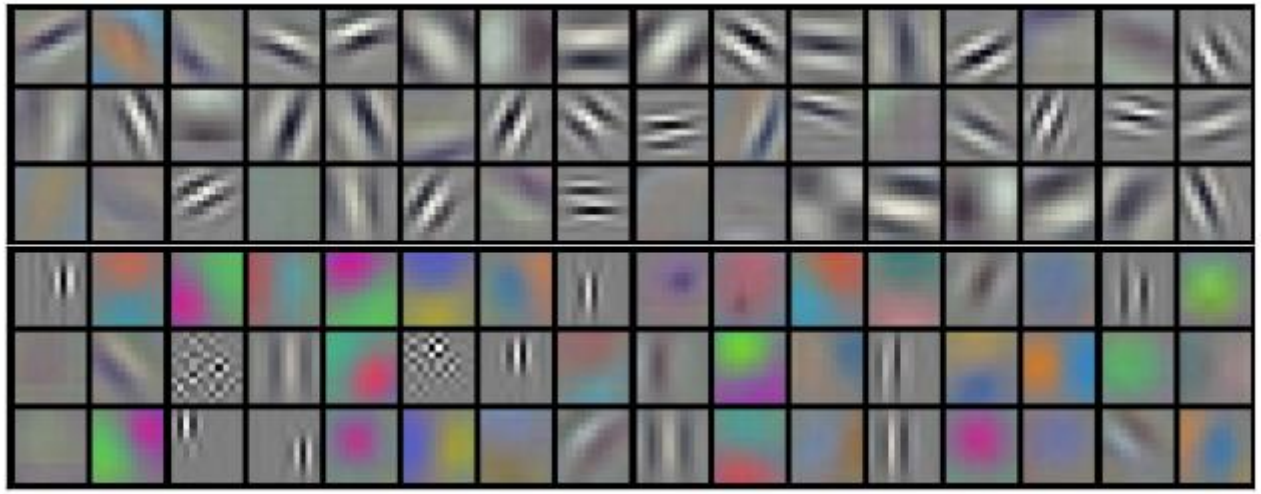
확률 값

분류된 class 결과 61

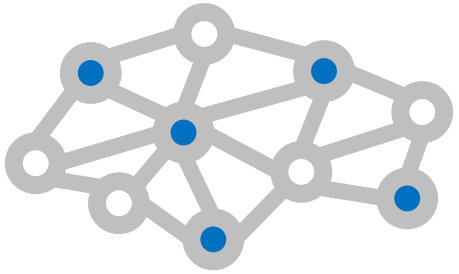


ImageNet Tutorial

- Convolution layer & blob visualization
  - Visualization of the first layer (conv1) in AlexNet
  - extract_features 프로젝트 빌드



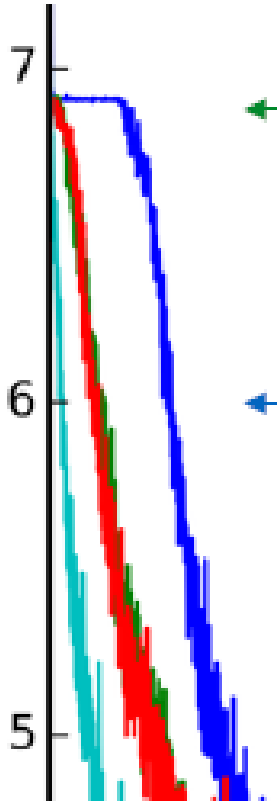
A Krizhevsky et al., NIPS 2012



Tips

- Caffe model zoo
  - https://github.com/BVLC/caffe/wiki/Model-Zoo
  - 여러 논문에 사용된 네트워크 구조가 올라와 있음
  - Network-in-Network (NIN) – 2013년 ImageNet 2위
  - vggNet – 2014년 ImageNet 1위
  - 등의 모델이 prototxt파일 형태로 있어 참고하기 좋음

Tips



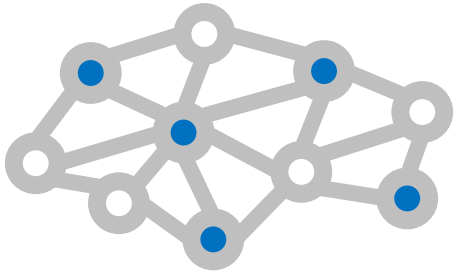
If loss is huge, something is not right.

Loss should start with random guess score log(1000)=6.91

An initial plateau usually means initialization scales are a bit off.

Good initializations make loss drop after a few iterations.

Slides from BVLC Caffe tutorial



Tips

- Loss가 일정 이상 줄어들지 않는다

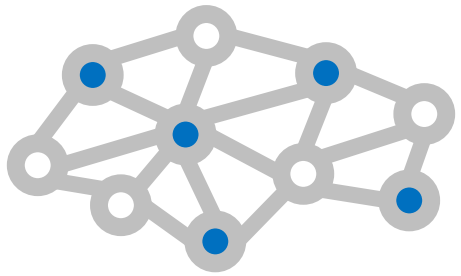
⇒CNN 구조를 더 복잡하게, filter를 더 많이 써본다.

- Training Loss가 줄어드는데 Test 성능은 좋아지지 않는다.

=> Overfitting의 가능성이 높으므로 CNN구조를 간단하게, filter 개수를 줄여본다.

- 초반에 loss가 줄어드는데 오래 걸린다

=> Initialization에 문제가 있다.



Tips

- 이외에도 많은 Caffe의 기능 및 옵션들이 있음
- 하지만 빠른 업데이트로 인해 최근에 추가된 기능들의 Documentation이 친절하지는 않음
- 최신의 가능한 Option들을 확인하려면 src/caffe/proto/caffe.proto 파일을 참고
- Github의 Pull request와 issue 및 google groups의 검색을 생활화

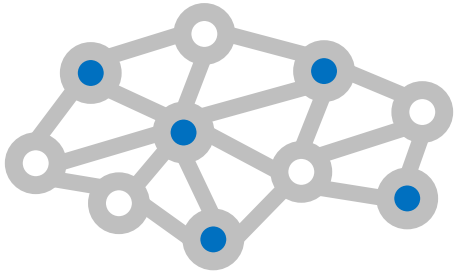
```
// Proto filename for the train net, possibly combined with one or more
// test nets.
optional string net = 24;
// Inline train net param, possibly combined with one or more test nets.
optional NetParameter net_param = 25;

optional string train_net = 1; // Proto filename for the train net.
repeated string test_net = 2; // Proto filenames for the test nets.
optional NetParameter train_net_param = 21; // Inline train net params.
repeated NetParameter test_net_param = 22; // Inline test net params.

// The states for the train/test nets. Must be unspecified or
// specified once per net.
//
// By default, all states will have solver = true;
// train_state will have phase = TRAIN,
// and all test_state's will have phase = TEST.
// Other defaults are set according to the NetState defaults.
optional NetState train_state = 26;
repeated NetState test_state = 27;

// The number of iterations for each test net.
repeated int32 test_iter = 3;

// The number of iterations between two testing phases.
optional int32 test_interval = 4 [default = 0];
optional bool test_compute_loss = 19 [default = false];
// If true, run an initial test pass before the first iteration,
// ensuring memory availability and printing the starting value of the loss.
optional bool test_initialization = 32 [default = true];
optional float base_lr = 5; // The base learning rate
// the number of iterations between displaying info. If display = 0, no info
// will be displayed.
optional int32 display = 6;
// Display the loss averaged over the last average_loss iterations
optional int32 average_loss = 33 [default = 1];
optional int32 max_iter = 7; // the maximum number of iterations
// accumulate gradients over `iter_size` x `batch_size` instances
optional int32 iter_size = 36 [default = 1];
optional string lr_policy = 8; // The learning rate decay policy.
optional float gamma = 9; // The parameter to compute the learning rate.
optional float power = 10; // The parameter to compute the learning rate.
optional float momentum = 11; // The momentum value.
```

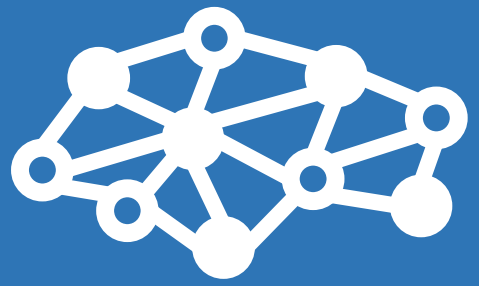


References

- Caffe github
  - https://github.com/BVLC/caffe
- Caffe intro & tutorial
  - http://caffe.berkeleyvision.org/
- Caffe-users Google Groups
  - https://groups.google.com/forum/#!forum/caffe-users
- Caffe BVLC tutorial slide
  - https://docs.google.com/presentation/d/1UeKXVgRvvxg9OUdh_UiC5G71UMscNPlvArsWER41PsU/edit#slide=id.gc2fcdcce7_216_0
- LeCun et al., 1998
  - LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
- Krizhevsky et al., 2012
  - Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.
- Dropout
  - Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

Thank You

Q&A



MIPAL aboratory

machine intelligence &pattern analysis