LeNet-5 Introduction

- a neural network architecture for handwritten and machine-printed character recognition -

2020 - 2021

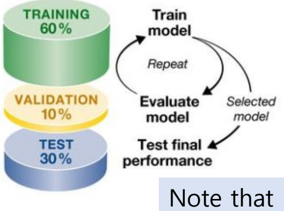
Ando Ki, Ph.D.
adki@future-ds.com

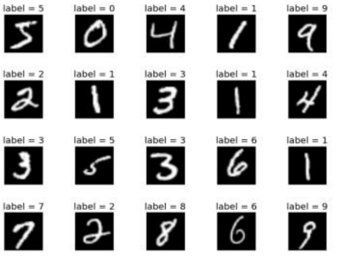
Table of contents

Copyright (c) Ando Ki

MNIST

- MNIST (Modified National Institute of Standards and Technology database)
  - Modified National Institute of Standards and Technology
  - Handwritten digits database
    - 10 classes: 0, 1, …, 9
    - training set: 60,000 training image
    - test set: 10,000 testing image





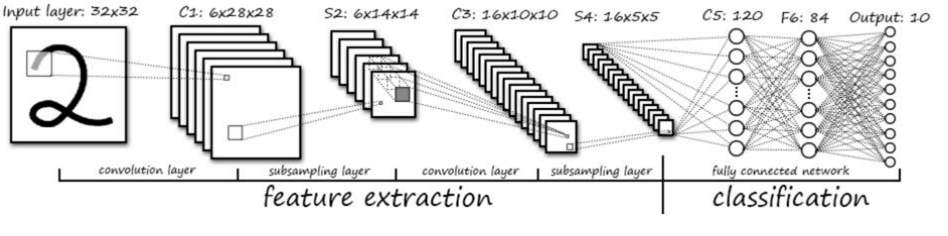
Note that MNIST images are 28x28.

Why background color is black?

Copyright (c) Ando Ki

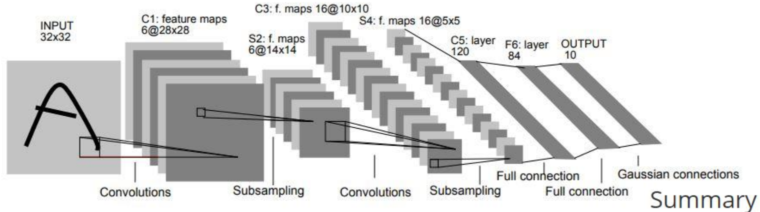
LeNet-5 for MNIST

- LeNet is one of the popular convolutional networks, and works well on digit classification tasks.
  - 1024 (32x32) inputs of black and white ➔ converted to floating number 0.0 ~ 1.0
  - 10 outputs representing digit 0 to 9



Copyright (c) Ando Ki

LeNet-5 for MNIST



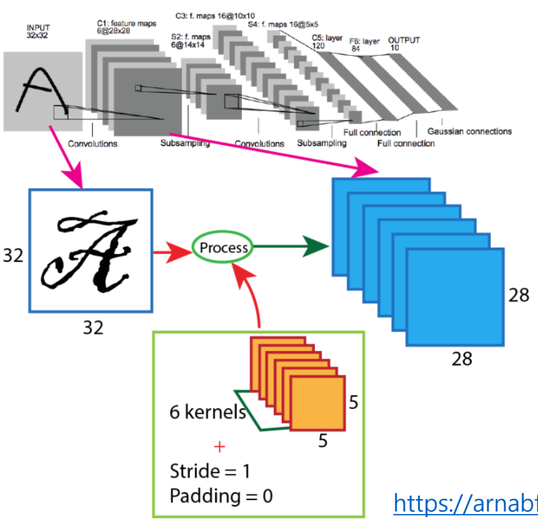
Original Image published in [LeCun et al., 1998]

Summary of LeNet-5 Architecture

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

tanh: hyperbolic tangent

LeNet-5 for MNIST: layer 1

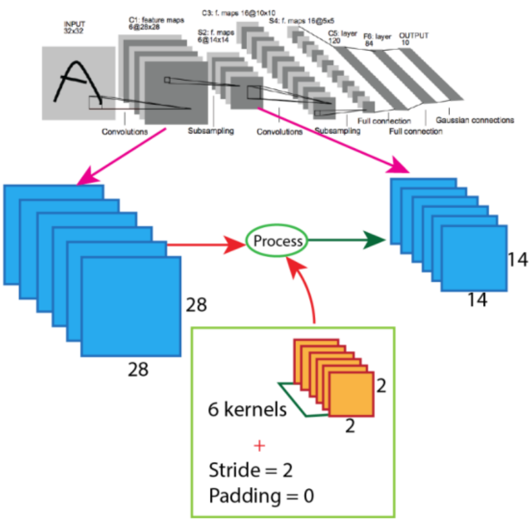


- 1st convolution layer
  - Input: 32x32 pixels (W=32)
  - Convolution filter: 6 kernels with 5x5 (K=5)
    - Parameters: weight+bias=1x5x5x6+6=156
  - Convolution padding: 0 (P=0)
  - Convolution: stride 1 (S=1)
  - Activation: ReLU (Tanh)
  - Results in: 6 features of 28x28

$$Output = \frac{W-K+2(P)}{S}+1 = \frac{32-5+2(0)}{1}+1 = 28$$

https://arnabfly.github.io/arnab_blog/lenet5/

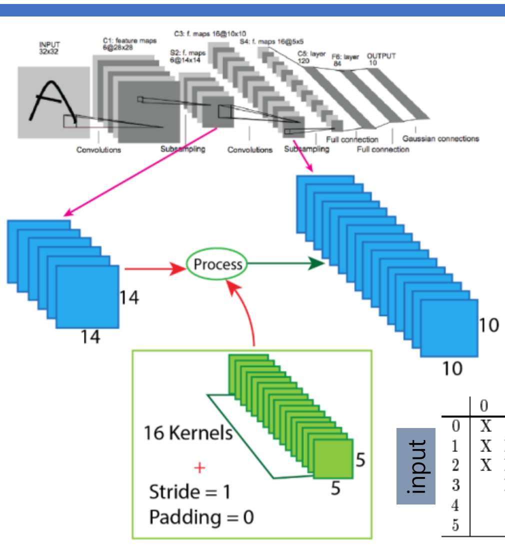
LeNet-5 for MNIST: layer 2



- 1^{st} pooling layer (sub-sampling)
  - Input: 6 features with 28x28 (W=28)
  - Max pooling filter: 2x2 (K=2)
  - Pooling padding: 0 (P=0)
  - Pooling: stride 2 (S=2)
    - It generates ½ number of elements
  - Activation: ReLU
  - Results in: 6 features of 14x14

$$Output = \frac{W - K + 2(P)}{S} + 1 = \frac{28 - 2 + 2(0)}{2} + 1 = 14$$

LeNet-5 for MNIST: layer 3



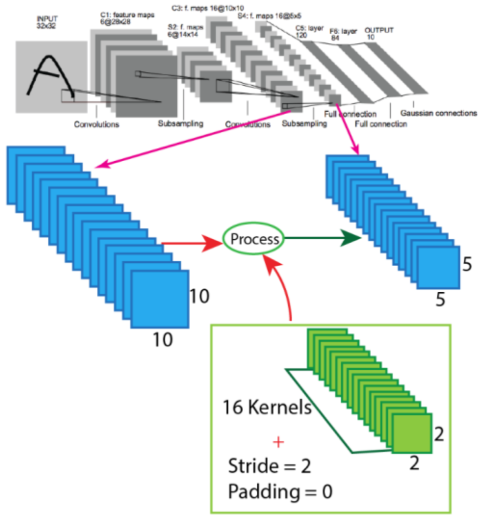
- 2^{nd} convolution
  - Input: 6 features with 14x14 pixels (W=14)
    - 6 kernels are used at the previous stage
  - Convolution filter: 16 kernels with 5x5 (K=5)
    - Parameters: 6x5x5x<u>16</u>+16=2,416
    - Reduction: 6x5x5x<u>10</u>+16=1,516
  - Convolution padding: 0 (P=0)
  - Convolution: stride 1 (S=1)
    - It generates the same number of elements
  - Activation: ReLU
  - Results in: 16 features of 10x10

feature maps

input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

only 10 out of 16 feature maps are connected to each kernel.

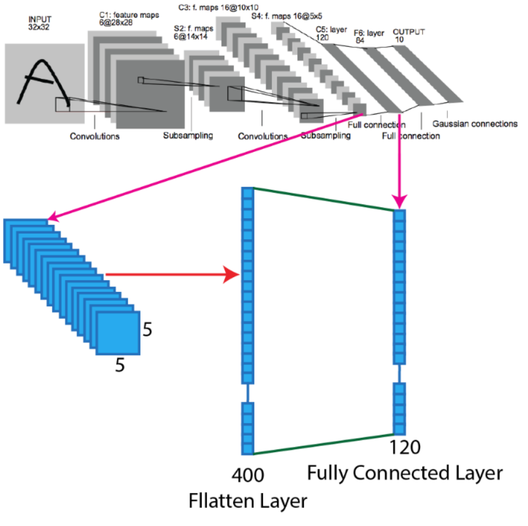
LeNet-5 for MNIST: layer 4



- 2^{nd} pooling
  - Input: 16 features with 10x10
  - Max pooling filter: 2x2
  - Pooling padding: 0
  - Pooling: stride 2
    - It generates ½ number of elements
  - Results in: 16 features of 5x5

Copyright (c) Ando Ki

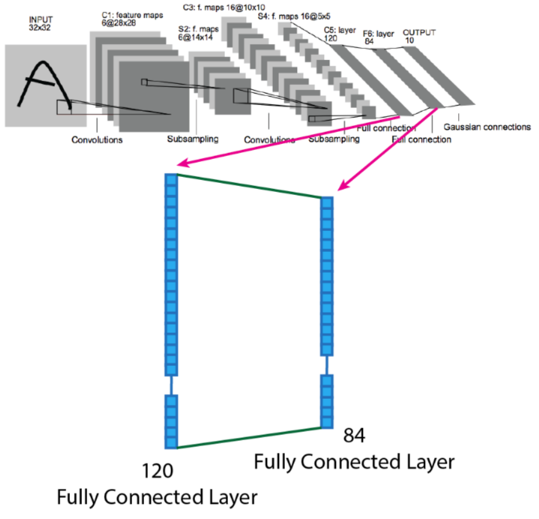
LeNet-5 for MNIST: layer 5



- fully connected layer for flatten
  - Input: 16 features with 5x5
  - Reshaping: 3-D array to 1-D vector
    - 16x5x5 ➔ 400
  - Output: 120
    - Neurons: 120
  - Parameters (weights+bias)
    - 400x120+120=48,120

Copyright (c) Ando Ki

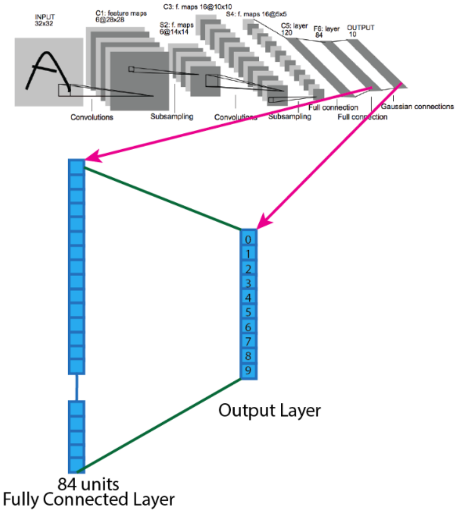
LeNet-5 for MNIST: layer 6



- fully connected layer for flatten
  - Input: 120 feature map
  - Output: 84
    - Neurons: 84
  - Parameters: 120x84+84=10,164

Copyright (c) Ando Ki

LeNet-5 for MNIST: layer 7



- fully connected layer for flatten
  - Input: 84 feature map
  - Output: 10
    - Neurons: 10
  - Parameters: 84x10+10=850

The output layer is composed of Euclidean Radial Basis Function unit (RBF), one for each class, with 84 inputs each. The outputs of each RBF unit i-th y is computed as follows.

$$y_i = \sum_j (x_j - w_{ij})^2.$$

Nowadays, softmax is used instead.

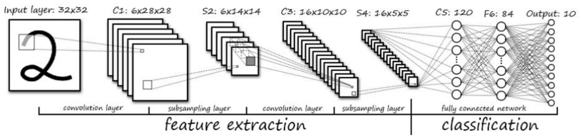
Copyright (c) Ando Ki

Summary of LeNet-5

Summary of LeNet-5 Architecture

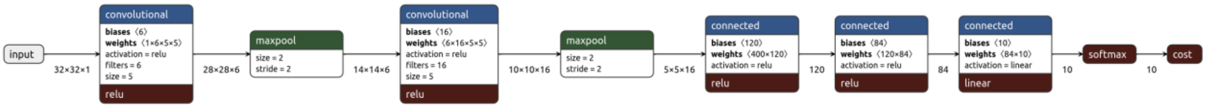
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416   (1516)
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```



Copyright (c) Ando Ki

Running LeNet-5: Darknet case



LeNet-5 configuration

```
[net]
batch=100
subdivisions=1
height=32
width=32
channels=1
momentum=0.9
decay=0.00005
max_crop=28

learning_rate=0.01
policy=poly
power=4
max_batches=500
```

```
angle=1
hue=1
saturation=1
exposure=1
aspect=1

[convolutional]
filters=6
size=5
stride=1
pad=0
activation=relu
```

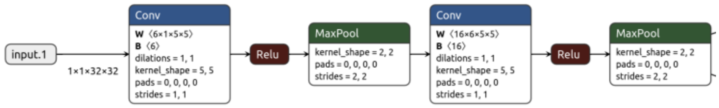
```
[maxpool]
size=2
stride=2

[convolutional]
filters=16
size=5
stride=1
pad=0
activation=relu

[maxpool]
size=2
stride=2
```

```
[connected]
output= 120
activation=relu

[connected]
output= 84
activation=relu

[connected]
output= 10
activation=linear
```

```
[softmax]
groups=1

[cost]
type=sse
```

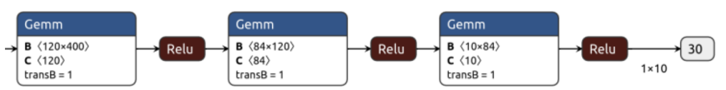
Copyright (c) Ando Ki

Running LeNet-5: PyTorch case (1/2)




```
from torch import nn
from torch.nn import Module
import torch.nn.functional as F

class Lenet5Model(Module):
    def __init__(self):
        super(Lenet5Model, self).__init__()
        self.conv1 = nn.Conv2d( in_channels=1, out_channels=6, kernel_size=(5,5)
                              , stride=1, padding=0, bias=True)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d( kernel_size=(2,2), stride=2 )
        self.conv2 = nn.Conv2d( in_channels=6, out_channels=16
                              , kernel_size=(5,5), stride=1
                              , padding=0
                              , bias=True)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d( kernel_size=(2,2), stride=2 )
```

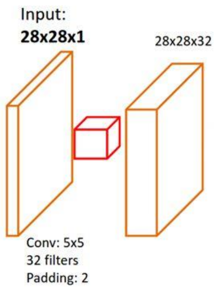

Running LeNet-5: PyTorch case (2/2)



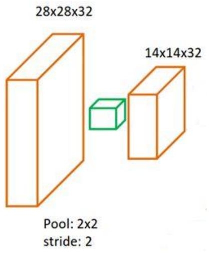

```
        self.fc1   = nn.Linear( in_features=16*5*5
                              , out_features=120
                              , bias=True)
        self.relu3 = nn.ReLU()
        self.fc2   = nn.Linear( in_features=120
                              , out_features=84
                              , bias=True)
        self.relu4 = nn.ReLU()
        self.fc3   = nn.Linear( in_features=84
                              , out_features=10
                              , bias=True)
        self.relu5 = nn.ReLU()
```

LeNet-5 for MNIST: layer

- 1^{st} convolution layer
  - Input: 28x28 pixels
  - Convolution filter: 32 kernels with 5x5
  - Convolution: stride 1
    - It generates the same number of elements
  - Results in: 32 features of 28x28



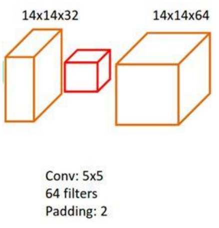
- 1^{st} pooling layer (sub-sampling)
  - Input: 32 features with 28x28
  - Max pooling filter: 5x5 (2x2 ?)
  - Convolution: stride 2
    - It generates ½ number of elements
  - Results in: 32 features of 14x14



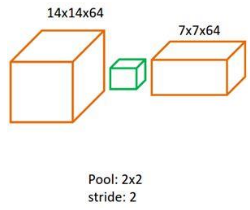
Copyright (c) Ando Ki

LeNet-5 for MNIST: layer

- 2^{nd} convolution
  - Input: 32 features with 14x14 pixels
    - 32 kernels are used at the previous stage
  - Convolution filter: 64 kernels with 5x5
  - Convolution: stride 1
    - It generates the same number of elements
  - Results in: 64 features of 14x14



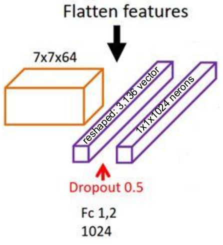
- 2^{nd} pooling
  - Input: 64 features with 14x14
  - Max pooling filter: 2x2
  - Convolution: stride 2
    - It generates ½ number of elements
  - Results in: 64 features of 7x7

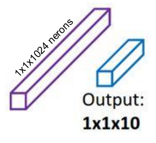


Copyright (c) Ando Ki

LeNet-5 for MNIST: layer

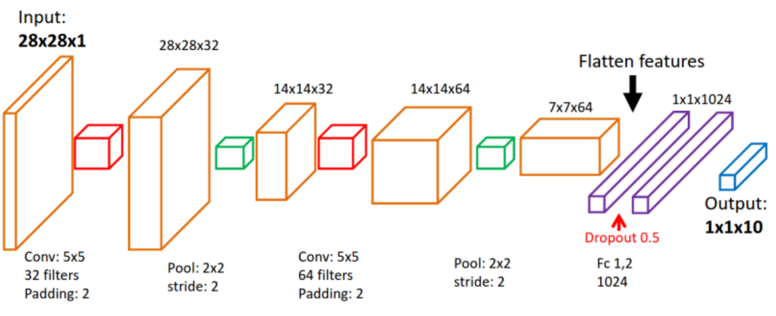
- fully connected layer
  - Input: 64 features with 7x7
  - Reshaping: 3-D array to 1-D vector
    - 64x7x7 ➔ 3,136
  - Neurons: 1024
- read-out layer
  - Input: 1024 neurons
  - Output: 10 classes

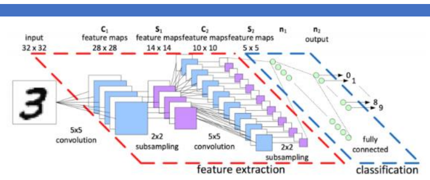


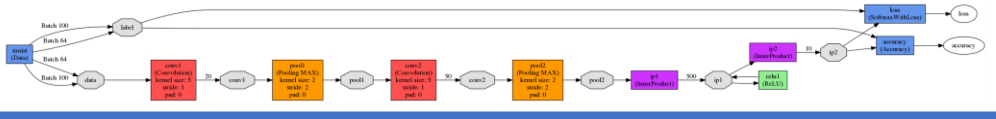


Copyright (c) Ando Ki

LeNet-5 for MNIST: all together



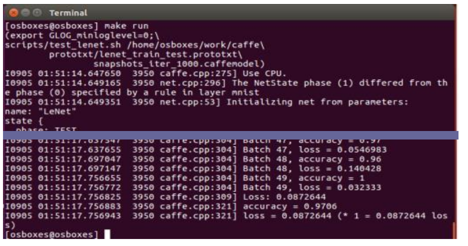




Copyright (c) Ando Ki

LeNet-5 for MNIST: running

- Steps (in details)
  - go to project directory
    - $ cd work/codes/caffe_v1-projects/mnist.LeNet
  - get dataset:
    - $ ./scripts/get_mnist.sh data
    - (ungip all in 'data' directory)
  - convert the dataset to Caffe data format
    - $ ./scripts/create_mnist.sh ${CAFFE_HOME} data
  - training
    - $ ./scripts/train_lenet.sh ${CAFFE_HOME} prototxt/lenet_solver.prototxt
  - running LeNet model with 'mnist_test_lmdb'
    - $ ./scripts/test_lenet.sh
- Step in simple
  - go to project directory
    - $ cd work/codes/caffe_v1-projects/mnist.LeNet
  - Run make
    - $ make cleanupall
    - $ make lmdb
    - $ make train
    - $ make test



Copyright (c) Ando Ki

LeNet-5 for MNIST: solver

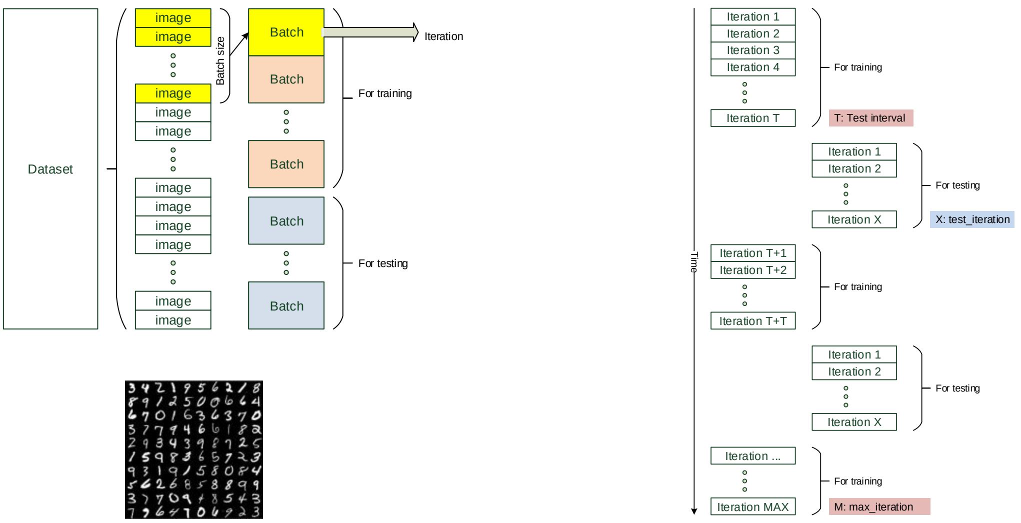
- net: network mode
- test_iter: iterations to test
- test_interval: interval between test
- base_lr: Learning Rate initial value
- display: iterations to show progress
- max_iter: max iterations for training.
- snapshot: iterations to store snapshot.
- solver_mode: CPU or GPU

```
# MNIST lenet_solver.prototxt
net: "lenet_train_test.prototxt"
test_iter: 100
test_interval: 500
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: “snapshots"
solver_mode: CPU
```

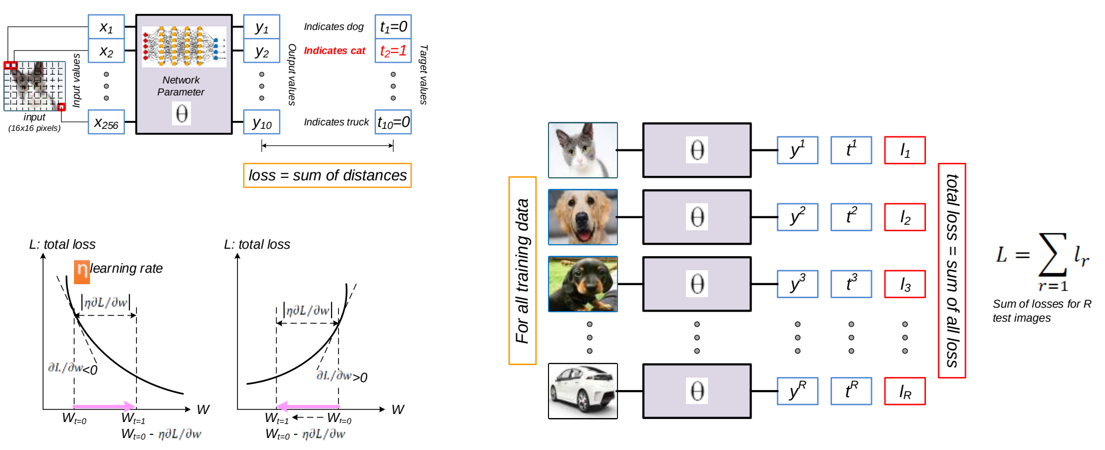
https://github.com/BVLC/caffe/wiki/Solver-Prototxt

Copyright (c) Ando Ki

LeNet-5 for MNIST: solver



LeNet-5 for MNIST: solver

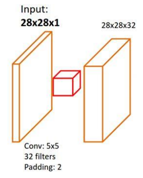


LeNet-5 for MNIST: net

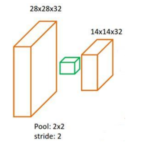
lenet_train_test.prototxt

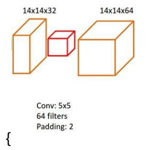
```
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
```

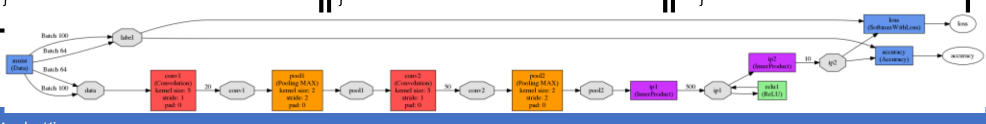
```
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```




```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
```

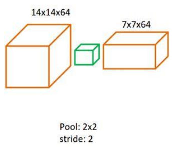


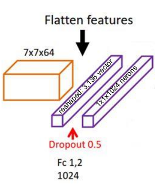




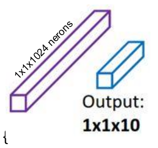
LeNet-5 for MNIST: net

```
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
```

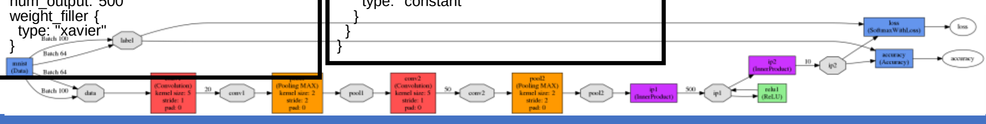





```
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

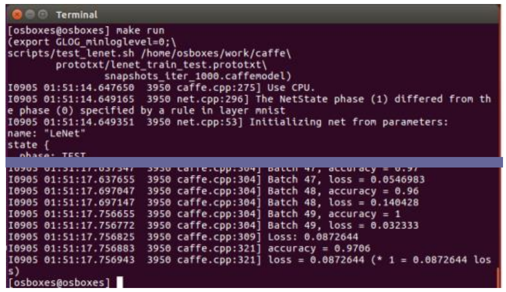



```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```



Running LeNet with Caffe

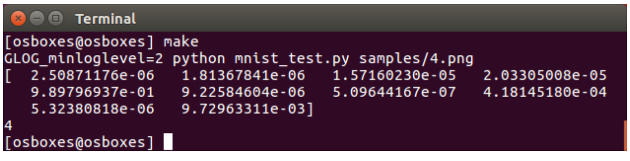
- This example is about LeNet
- Make sure 'work/caffe' is ready
  - see the pervious slides
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes.caffe/mnist.LeNet
  - Step 2: check network
    - [user@host] make draw
    - [user@host] fim lenet_train_test.png
  - Step 3: make data (convert data)
    - [user@host] make lmdb
  - Step 4: run train (it takes time)
    - [user@host] make train
  - Step 5: run loss graph
    - [user@host] make plot
  - Step 6: run test
    - [user@host] make test
  - Step 7: run deployment (inference)
    - [user@host] make deploy

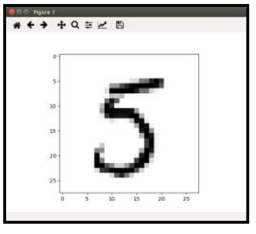


use 'display' for Ubuntu, 'fim' for Raspbian' to display image.

Run inference with sample image

- Go to 'mnist.LeNet' directory
  - $ cd .../codes/caffe_v1-project/mnist.LeNet
- Run make
  - $ make deploy
- Note that LeNet uses inverted image, i.e., background should be black.





Deploy prototxt (1/2)

- Refer to 'lenet_deploy.prototxt' under 'prototxt' directory.

```
###################### Remove the data layer
#layer {
#  name: "mnist"
#  type: "Data"
#  top: "data"
#  top: "label"
#  include {
#    phase: TRAIN
#  }
#  transform_param {
#    scale: 0.00390625
#  }
#  data_param {
#    source: "data/mnist_train_lmdb"
#    batch_size: 64
#    backend: LMDB
#  }
#}
###################### Remove the label layer
#layer {
#  name: "mnist"
#  type: "Data"
#  top: "data"
#  top: "label"
#  include {
#    phase: TEST
#  }
#  transform_param {
#    scale: 0.00390625
#  }
#  data_param {
#    source: "data/mnist_test_lmdb"
#    batch_size: 100
#    backend: LMDB
#  }
#}
```

Remove

Remove

```
###################### Add a new layer to accept data without label
### It define the name and shapes of the input blobs.
#       shape: { dim: 1    # batchsize (how many images/samples are fed through the network in paralle)
#               dim: 1    # number of channels; 1 means greay, not RGB
#               dim: 28   # heigh of data, i.e., pixels (MNIST data is 28x28)
#               dim: 28   # width of data, i.e., pixels (MNIST data is 28x28)
#       }
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
     shape: { dim: 1
              dim: 1
              dim: 28
              dim: 28
     }
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  ......
  other hidden layers remains
```

Add

Deploy prototxt (2/2)

- Refer to 'lenet_deploy.prototxt' under 'prototxt' directory.

```
...
... ...
  }
}
###################### Remove the layers depending upon ata labels
###################### accordingly remove layer that uses 'data' as bottom
#layer {
#  name: "accuracy"
#  type: "Accuracy"
#  bottom: "ip2"
#  bottom: "label"
#  top: "accuracy"
#  include {
#    phase: TEST
#  }
#}
#layer {
#  name: "loss"
#  type: "SoftmaxWithLoss"
#  bottom: "ip2"
#  bottom: "label"
#  top: "loss"
#}
############# Add a new layer to the end of this network to produce a Softmax output
layer {
  name: "loss"
  type: "Softmax"
  bottom: "ip2"
  top: "loss"
}
```

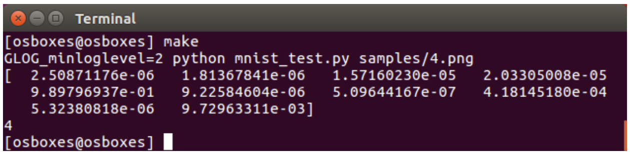
Remove

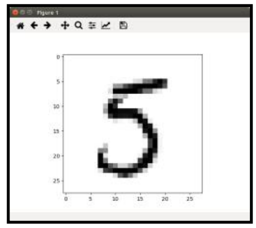
Remove

Add

Run inference with sample image (another way)

- Go to 'mnist.LeNet.python' directory
  - $ cd .../codes/caffe_v1-project/mnist.LeNet.python
- Run make
  - $ make
- Note that LeNet uses inverted image, i.e., background should be black.





Copyright (c) Ando Ki

Caffe Python interface for LeNet

mnist_test.py

```
import os
os.environ['GLOG_minloglevel']='2'

import caffe
import numpy as np
import cv2
import sys
import Image

caffe_home = os.environ["CAFFE_ROOT"];
model = caffe_home + '/examples/mnist/lenet.prototxt';
weights = '../mnist.LeNet/snapshots_iter_1000.caffemodel';
net = caffe.Net(model,weights,caffe.TEST);
caffe.set_mode_cpu()

img = cv2.imread(sys.argv[1],0)
if img.shape != [28,28]:
   img2 = cv2.resize(img,(28,28))
   img = img2.reshape(28,28,-1);
else:
   img = img.reshape(28,28,-1);

img = 1.0 - img/255.0

out = net.forward_all(data=np.asarray([img.transpose(2,0,1)]))

print out['prob'][0]
print out['prob'][0].argmax()
```

Using pycaffe

This may not need.

Revert image and normalize it to 0~1

Inference

Get the highest probability one

Copyright (c) Ando Ki

Running LeNet-5: TensorFlow case

Running LeNet-5: Keras case

References

- Yann LeCun and et.al., Gradient-Based Learning Applied to Document Recognition, Proc. of the IEEE, Nov. 1998.
- Break Down Lenet-5
  - https://arnabfly.github.io/arnab_blog/lenet5/
- LeNet-5 – A Classic CNN Architecture
  - https://www.datasciencecentral.com/profiles/blogs/lenet-5-a-classic-cnn-architecture
  - https://engmrk.com/lenet-5-a-classic-cnn-architecture/

Copyright (c) Ando Ki