

# Caffe V1 Examples

- LeNet and YOLO -

2019 – 2020

Ando Ki, Ph.D.

[adki@future-ds.com](mailto:adki@future-ds.com)

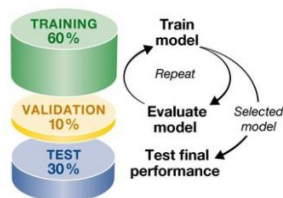
## LeNet Example

- LeNet-5 for MNIST
- LeNet-5 for MNIST: layer
- LeNet-5 for MNIST: all together
- LeNet-5 for MNIST: running
- LeNet-5 for MNIST: solver
- LeNet-5 for MNIST: net
- Running LeNet with Caffe
- Run inference with sample image
- Deploy prototxt
- Caffe Python interface for LeNet

# LeNet-5 for MNIST

## ■ MNIST dataset

- 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

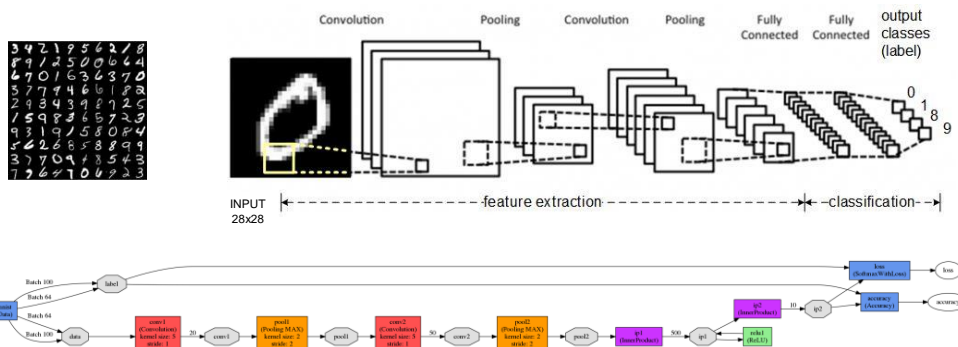


3

# LeNet-5 for MNIST

## ■ LeNet is one of the popular convolutional networks, and works well on digit classification tasks.

- ⇒ 784 (28x28) inputs of black and white → converted to floating number 0.0 ~ 1.0
- ⇒ 10 outputs representing digit 0 to 9

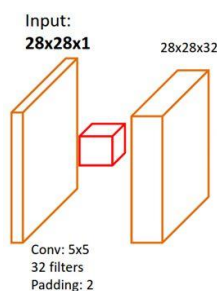


4

## LeNet-5 for MNIST: layer

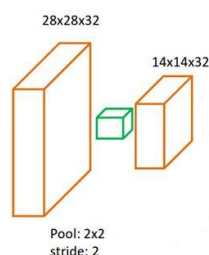
### 1<sup>st</sup> convolution layer

- ▶ 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<sup>st</sup> pooling layer (sub-sampling)

- ▶ Input: 32 features with 28x28
- ▶ Max pooling filter: 5x5 (2x2 ?)
- ▶ Convolution: stride 2
  - ☞ It generates  $\frac{1}{2}$  number of elements
- ▶ Results in: 32 features of 14x14

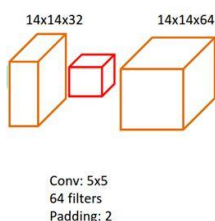


5

## LeNet-5 for MNIST: layer

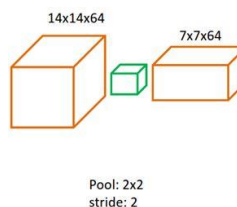
### 2<sup>nd</sup> convolution

- ▶ 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<sup>nd</sup> pooling

- ▶ Input: 64 features with 14x14
- ▶ Max pooling filter: 2x2
- ▶ Convolution: stride 2
  - ☞ It generates  $\frac{1}{2}$  number of elements
- ▶ Results in: 64 features of 7x7



6



## 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`
  - (ungzip 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 cleanall`
  - `$ make lmdb`
  - `$ make train`
  - `$ make test`

```

[osboxes@osboxes] make run
[export GLOG_minloglevel=0]
[scripts/test_lenet.sh /home/osboxes/work/caffe/prototxt/lenet_train_test.prototxt]
[0905 01:51:14.047650 3950 caffe.cpp:275] Use CPU.
[0905 01:51:14.049351 3950 net.cpp:290] The NetState phase (1) differed from the phase (0) specified by a rule in layer mnist
[0905 01:51:14.049351 3950 net.cpp:53] Initializing net from parameters:
name: "lenet"
state {
  phase: "test"
}
[0905 01:51:17.037341 3950 caffe.cpp:304] Batch 47, accuracy = 0.957
[0905 01:51:17.037655 3950 caffe.cpp:304] Batch 47, loss = 0.0546983
[0905 01:51:17.037647 3950 caffe.cpp:304] Batch 48, accuracy = 0.956
[0905 01:51:17.037147 3950 caffe.cpp:304] Batch 48, loss = 0.140428
[0905 01:51:17.756855 3950 caffe.cpp:304] Batch 49, accuracy = 1
[0905 01:51:17.756772 3950 caffe.cpp:304] Batch 49, loss = 0.032333
[0905 01:51:17.756825 3950 caffe.cpp:309] Loss: 0.0872644
[0905 01:51:17.756883 3950 caffe.cpp:321] accuracy = 0.9706
[0905 01:51:17.756943 3950 caffe.cpp:321] loss = 0.0872644 (* 1 = 0.0872644 loss)
[osboxes@osboxes]

```

9

## 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

```

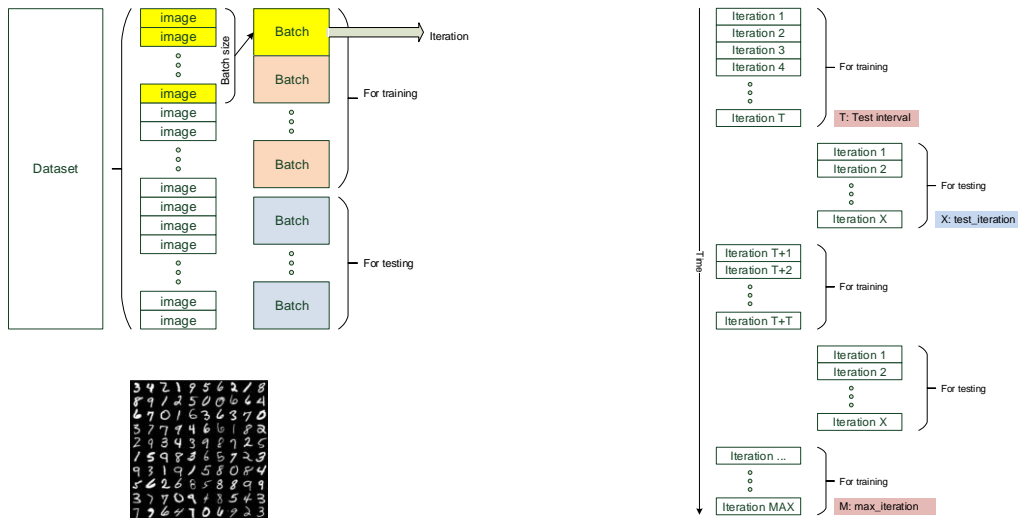
# MNIST lenet_solver.prototxt
net: "lenet_train_test.prototxt"
test_iter: 100
test_interval: 500
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"
gamma: 0.0001
power: 0.75
display: 100
max_iter: 10000
snapshot: 5000
snapshot_prefix: "snapshots"
solver_mode: CPU

```

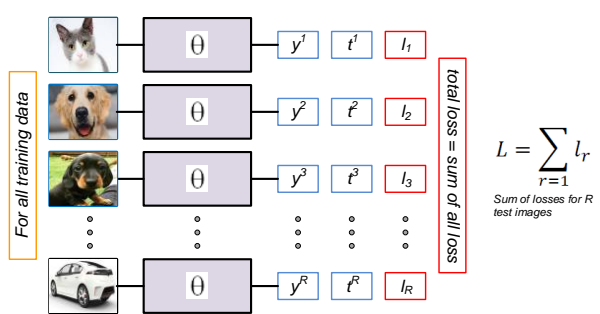
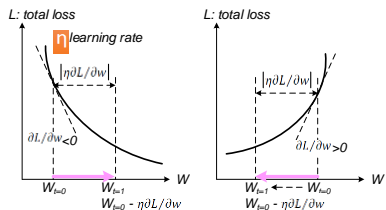
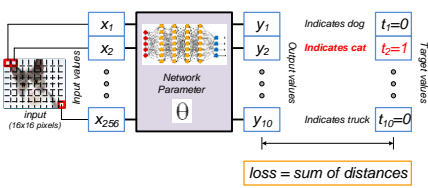
<https://github.com/BVLC/caffe/wiki/Solver-Prototxt>

10

# LeNet-5 for MNIST: solver



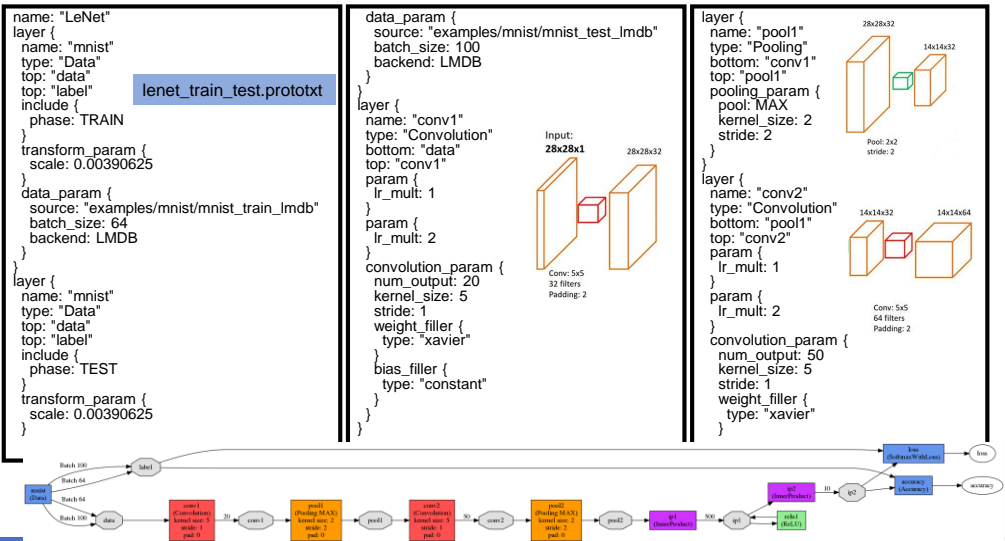
# LeNet-5 for MNIST: solver



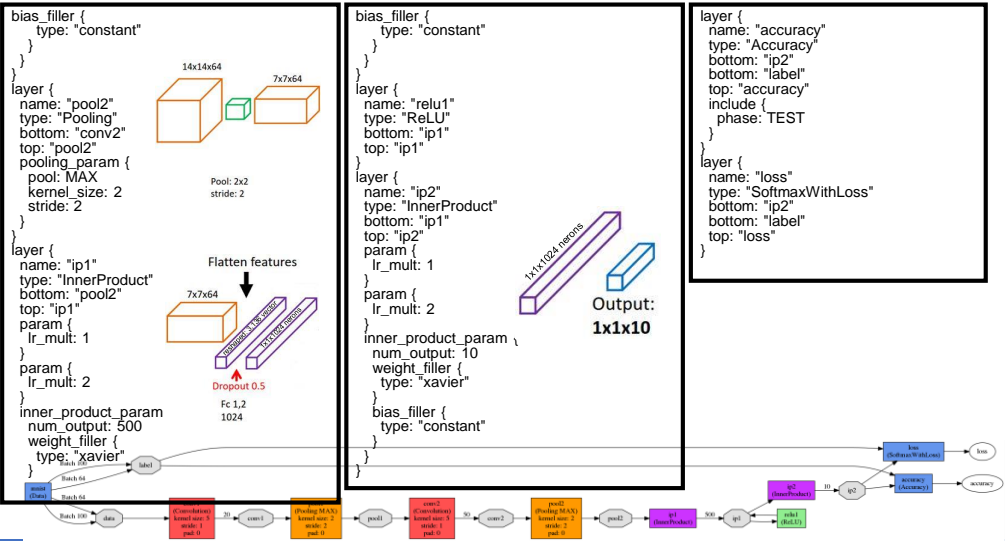
$$L = \sum_{r=1}^R l_r$$

Sum of losses for  $R$  test images

# LeNet-5 for MNIST: net

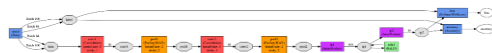


# LeNet-5 for MNIST: net



## 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



```

[osboxes@osboxes] make run
(export GLOG_minloglevel=0)
scripts/test_lenet.sh /home/osboxes/work/caffe/
prototxt/lenet_train_test.prototxt\
snapshots_iter_1000.caffemodel)
10905 01:51:14.647650 3950 caffe.cpp:275] Use CPU.
10905 01:51:14.649165 3950 net.cpp:296] The NetState phase (1) differed from th
e phase (0) specified by a rule in layer mnist
10905 01:51:14.649351 3950 net.cpp:53] Initializing net from parameters:
name: "LeNet"
state {
  TRAIN - TEST
10905 01:51:17.037341 3950 caffe.cpp:304] Batch 47, accuracy = 0.97
10905 01:51:17.637655 3950 caffe.cpp:304] Batch 47, loss = 0.0540983
10905 01:51:17.697047 3950 caffe.cpp:304] Batch 48, accuracy = 0.96
10905 01:51:17.697147 3950 caffe.cpp:304] Batch 48, loss = 0.140428
10905 01:51:17.756655 3950 caffe.cpp:304] Batch 49, accuracy = 1
10905 01:51:17.756772 3950 caffe.cpp:304] Batch 49, loss = 0.032333
10905 01:51:17.756825 3950 caffe.cpp:309] Loss: 0.0872644
10905 01:51:17.756833 3950 caffe.cpp:321] accuracy = 0.9766
10905 01:51:17.756943 3950 caffe.cpp:321] loss = 0.0872644 (* 1 = 0.0872644 los
s)
[osboxes@osboxes]

```

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

15

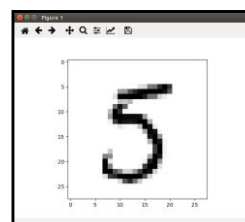
## Run inference with sample image

- 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.

```

[osboxes@osboxes] make
GLOG_minloglevel=2 python mnist_test.py samples/4.png
[ 2.50871176e-06  1.81367841e-06  1.57160230e-05  2.03305008e-05
 9.89796937e-01  9.22584604e-06  5.09644167e-07  4.18145180e-04
 5.32380818e-06  9.72963311e-03]
4
[osboxes@osboxes]

```



16



## Deploy prototxt (1/2)

- Refer to 'lenet\_deploy.prototxt' under 'prototxt' directory.

<pre>##### 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 #  } #}</pre>	<pre>##### 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 parallel) #         dim: 28 # height 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: 28 #              dim: 28 #            } #   } # }  layer {   name: "conv1"   type: "Convolution"   bottom: "data"   top: "conv1"   param {     lr_mult: 1   }   param {     lr_mult: 2   } }  ##### other hidden layers remains</pre>
---	--

17

## Deploy prototxt (2/2)

- Refer to 'lenet\_deploy.prototxt' under 'prototxt' directory.

```
...
...
}
}

##### Remove the layers depending upon data 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"
}
```

18

## Run inference with sample image (another way)

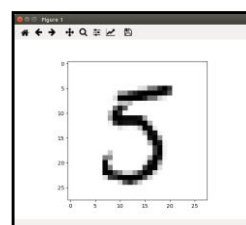
- Go to 'mnist.LeNet.python' directory

- \$ cd ../codes/caffe\_v1-project/mnist.LeNet.python

- Run make

- \$ make

```
Terminal
[osboxes@osboxes] make
GLOG_minloglevel=2 python mnist_test.py samples/4.png
[ 2.50871176e-06  1.81367841e-06  1.57160230e-05  2.03305008e-05
 9.89796937e-01  9.22584604e-06  5.09644167e-07  4.18145180e-04
 5.32380818e-06  9.72963311e-03]
4
[osboxes@osboxes]
```



19

## Caffe Python interface for LeNet

```
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

20

# YOLO

- Darknet: The framework for YOLO
- Darknet example for detector
- Network configuration file
- YOLO
- YOLO model
- YOLO network
- YOLO on Caffe

21

## Darknet: The framework for YOLO

- Darknet is an open source neural network framework written in C and CUDA (Compute Unified Device Architecture) supporting CPU (Central Processing Unit) and GPU (Graphical Processing Unit) computation.
  - ▶ <https://github.com/pjreddie/darknet>
  - ▶ <https://pjreddie.com/darknet/>
  - ▶ <https://groups.google.com/forum/#!forum/darknet>
  - ▶ Site: <https://pjreddie.com/darknet/>
  - ▶ GitHub : <https://github.com/pjreddie/darknet>

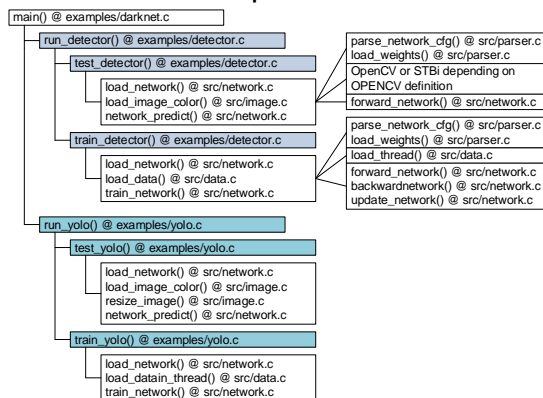


"Darknet: Open Source Neural Networks in C", Joseph Redmon, <http://pjreddie.com/darknet>, 2013-2016.

22

## Darknet example for detector

### ■ Have a look at 'darknet/examples/darknet.c' file



23

## Network configuration file

### ■ Nodes

- ▶ [shortcut]
- ▶ [crop]
- ▶ [cost]
- ▶ [detection]
- ▶ [region]
- ▶ [local]
- ▶ [conv] or [convolutional]
- ▶ [deconv] or [deconvolutional]
- ▶ [activation]
- ▶ [net] or [network]
- ▶ [crnn]
- ▶ [gru]
- ▶ [lstm]
- ▶ [rnn]
- ▶ [conn] or [connected]
- ▶ [max] or [maxpool]
- ▶ [reorg]

<https://github.com/cvjenal/darknet/blob/master/cfg/yolo.cfg>

- ▶ [avg] or [avgpool]
- ▶ [dropout]
- ▶ [ln] or [normalization]
- ▶ [batchnorm]
- ▶ [soft] or [softmax]
- ▶ [route]

```
[net]
batch=1
subdivisions=1
width=416
height=416
channels=3
```

```
[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky
```

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

24

## Network configuration file

- [net] node
  - ▶ batch: how many images are in each batch to average the loss over?
  - ▶ subdivisions: into how many sub-batches shall each batch be divided to handle images in each sub-batch in parallel?
  - ▶ height, width: input size of the network
  - ▶ channels: number of components, e.g., color components
  - ▶ momentum: learning parameters
  - ▶ learning\_rate: base learning rate
  - ▶ policy: change learning rate after the corresponding steps
  - ▶ steps: need to have as many steps as scale
  - ▶ scales: re-scale the current learning rate by the corresponding factor once the number of steps is reached
  - ▶ max\_batches: max number of "iterations"
  - ▶ i\_snapshot\_iteration: snapshot the learned weights after every k "iterations"
- [convolutional] node
  - ▶ filters: number of filters, i.e., kernels
  - ▶ size: size of filter, e.g., 3 means 3x3 filter
  - ▶ stride: number of stride
  - ▶ pad: number of padding, e.g., 1
  - ▶ activation: specify activation function
- [maxpool] node
  - ▶ size: size of filter
  - ▶ stride: number of stride
- [connected] node
  - ▶ output: number of output of fully connected network
  - ▶ activation: activation function
- [detection] node or [region] node
  - ▶ classes: number of classes, e.g., 20 for pascal voc (l.classes)
  - ▶ coords: bounding boxes -> 4 parameters (l.n)
  - ▶ side: number of cell in x and y direction
  - ▶ num: number of predicted boxes per cell

25

## Network configuration file

- First node should be [net]

```

[net]
width=416
height=416
channels=3

[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2048
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4096
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=8192
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=16384
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=32768
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=65536
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=131072
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=262144
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=524288
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1048576
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2097152
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4194304
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=8388608
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=16777216
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=33554432
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=67108864
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=134217728
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=268435456
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=536870912
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1073741824
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2147483648
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4294967296
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=8589934592
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=17179869184
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=34359738368
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=68719476736
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=137438953472
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=274877906944
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=549755813888
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1099511627776
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2199023255552
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4398046511104
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=8796093022208
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=17592186044416
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=35184372088832
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=70368744177664
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=140737488355328
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=281474976710656
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=562949953421312
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1125899906842624
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2251799813685248
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4503599627370496
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=9007199254740992
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=18014398509481984
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=36028797018963968
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=72057594037927936
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=144115188075855872
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=288230376151711744
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=576460752303423488
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1152921504606846976
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2305843009213693952
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4611686018427387904
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=9223372036854775808
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=18446744073709551616
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=36893488147419103232
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=73786976294838206464
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=147573952589676412928
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=295147905179352825856
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=590295810358705651712
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1180591620717411303424
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2361183241434822606848
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4722366482869645213696
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=9444732965739290427392
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=18889465931478580854784
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=37778931862957161709568
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=75557863725914323419136
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=151115727451828646838272
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=302231454903657293676544
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=604462909807314587353088
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1208925819614629174706176
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2417851639229258349412352
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4835703278458516698824704
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=9671406556917033397649408
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=19342813113834066795298816
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=38685626227668133590597632
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=77371252455336267181195264
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=154742504910672534362390528
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=309485009821345068724781056
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=618970019642690137449562112
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1237940039285380274899124224
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2475880078570760549798248448
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=4951760157141521099596496896
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=9903520314283042199192993792
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=19807040628566084398385987584
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=39614081257132168796771975168
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=79228162514264337593543950336
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=158456325028528675187087900672
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=316912650057057350374175801344
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=633825300114114700748351602688
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1267650600228229401496703205376
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2535301200456458802993406410752
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5070602400912917605986812821504
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=10141204801825835211973625643008
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=20282409603651670423947251286016
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=40564819207303340847894502572032
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=81129638414606681695789005144064
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=162259276829213363391578010288128
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=324518553658426726783156020576256
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=649037107316853453566312041152512
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1298074214633706907132624082305024
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2596148429267413814265248164610048
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5192296858534827628530496329220096
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=10384593717069655257060992658440192
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=20769187434139310514121985316880384
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=41538374868278621028243970633760768
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=83076749736557242056487941267521536
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=166153499473114484112975882535043072
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=332306998946228968225951765070086144
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=664613997892457936451903530140172288
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1329227995784915872903807060280344576
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2658455991569831745807614120560689152
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5316911983139663491615228241121378304
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=10633823966279326983230456482242756608
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=21267647932558653966460912964485513216
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=42535295865117307932921825928971026432
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=85070591730234615865843651857942052864
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=170141183460469231731687303715884105728
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=340282366920938463463374607431768211456
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=680564733841876926926749214863536422912
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1361129467683753853853498429727072845824
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2722258935367507707706996859454145691648
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5444517870735015415413993718908291383296
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=10889035741470030830827987437816582766592
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=21778071482940061661655974875633165533184
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=43556142965880123323311949751266331066368
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=87112285931760246646623899502532662132736
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=174224571863520493293247799005065324265472
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=348449143727040986586495598010130648530944
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=696898287454081973172991196020261297061888
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1393796574908163946345982392040522594123776
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2787593149816327892691964784081045188247552
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5575186299632655785383929568162090376495104
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=11150372599265311570767859136324180752990208
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=22300745198530623141535718272648361505980416
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=44601490397061246283071436545296723011960832
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=89202980794122492566142873090593446023921664
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=178405961588244985132285746181186892047843328
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=356811923176489970264571492362373784095686656
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=713623846352979940529142984724747568191373312
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1427247692705959881058285969449495136382746624
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2854495385411919762116571938898990272765493248
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5708990770823839524233143877797980545530986496
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=11417981541647679048466287755595961091061972992
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=22835963083295358096932575511191922182123945984
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=45671926166590716193865151022383844364247891968
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=91343852333181432387730302044767688728495783936
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=182687704666362864775460604089535377456991567872
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=365375409332725729550921208179070754913983135744
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=730750818665451459101842416358141509827966271488
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1461501637330902918203684832716283019655932542976
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2923003274661805836407369665432566039311865085952
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5846006549323611672814739330865132078623730171904
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=11692013098647223345629478661730264157247460343808
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=23384026197294446691258957323460528314494920687616
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=46768052394588893382517914646921056628989841375232
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=93536104789177786765035829293842113257979682750464
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=187072209578355573530071658587684226515959365500928
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=374144419156711147060143317175368453031918731001856
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=748288838313422294120286634350736906063837462003712
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1496577676626844588240573268701473812127674924007424
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=2993155353253689176481146537402947624255349848014848
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=5986310706507378352962293074805895248510699696029696
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=11972621413014756705924586149611790497021399392059392
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=23945242826029513411849172299223580994042798784118784
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=47890485652059026823698344598447161988085597568237568
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=95780971304118053647396689196894323976171195136475136
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=191561942608236107294793378393788647952342390272950272
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=383123885216472214589586756787577295904684780545900544
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=766247770432944429179173513575154591809369561091801088
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1532495540865888858358347027150309183618739122183602176
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=3064991081731777716716694054300618367237478244367204352
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=6129982163463555433433388108601236734474956488734408704
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=12259964326927110866866776217202473468949912977468817408
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=24519928653854221733733552434404946937899825954937634816
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=49039857307708443467467104868809893875799651909875269632
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=98079714615416886934934209737619787751599303819750539264
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=196159429230833773869868419475239575503198607639501078528
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=392318858461667547739736838950479151006397215279002157056
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=784637716923335095479473677900958302012794430558004314112
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1569275433846670190958947355801916604025588861116008628224
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=3138550867693340381917894711603833208051177722232017256448
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=6277101735386680763835789423207666416102
```

# Network configuration file

■ First node should be [net]

tiny-yolo-voc.cfg

layer	filters	size	input	output
0 conv	16	3 x 3 / 1	416 x 416 x 3	16 x 416 x 416
1 max	2	2 x 2 / 2	416 x 416 x 16	16 x 208 x 208
2 conv	32	3 x 3 / 1	208 x 208 x 16	32 x 208 x 208
3 max	2	2 x 2 / 2	208 x 208 x 32	32 x 104 x 104
4 conv	64	3 x 3 / 1	104 x 104 x 32	64 x 104 x 104
5 max	2	2 x 2 / 2	104 x 104 x 64	64 x 52 x 52
6 conv	128	3 x 3 / 1	52 x 52 x 64	128 x 52 x 52
7 max	2	2 x 2 / 2	52 x 52 x 128	128 x 26 x 26
8 conv	256	3 x 3 / 1	26 x 26 x 128	256 x 26 x 26
9 max	2	2 x 2 / 2	26 x 26 x 256	256 x 13 x 13
10 conv	512	3 x 3 / 1	13 x 13 x 256	512 x 13 x 13
11 max	2	2 x 2 / 1	13 x 13 x 512	512 x 13 x 13
12 conv	1024	3 x 3 / 1	13 x 13 x 512	1024 x 13 x 13
13 conv	1024	3 x 3 / 1	13 x 13 x 1024	1024 x 13 x 13
14 conv	125	1 x 1 / 1	13 x 13 x 1024	125 x 13 x 13
15 detection				

```
[net]
batch=64
subdivisions=8
width=416
height=416
channels=3

learning_rate=0.001
max_batches = 40200
policy=steps
steps=-
1,100,20000,30000
scales=.1,10,.1,.1

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=125
size=1
stride=1
pad=1
activation=linear

[region]
anchors =
1.08,1.19,
3.42,4.41,
6.63,11.38,
9.42,5.11,
16.62,10.52
bias_match=1
classes=20
coords=4
batch_normalize=1
num=5
softmax=1
jitter=.2
rescore=1
object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1
absolute=1
thresh = .6
random=1
```

# YOLO

■ YOLO (You Only Look Once)

- ▶ An implementation of object-detection using Darknet.
- ▶ YOLO: <https://pjreddie.com/darknet/yolo/> → YOLO3
- ▶ YOLO (darknet): <https://pjreddie.com/darknet/yolov1/>
- ▶ YOLOv2 (darknet): <https://pjreddie.com/darknet/yolo/>
- ▶ YOLO (caffe): <https://github.com/xingwangsfu/caffe-yolo>
- ▶ YOLO (TensorFlow: Train+Test): <https://github.com/thtrieu/darkflow>
- ▶ YOLO (TensorFlow: Test): [https://github.com/gliese581gg/YOLO\\_tensorflow](https://github.com/gliese581gg/YOLO_tensorflow)



■ Why it is called YOLO

- ▶ A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.

"You Only Look Once: Unified, Real-Time Object Detection", Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.



## YOLO on Caffe

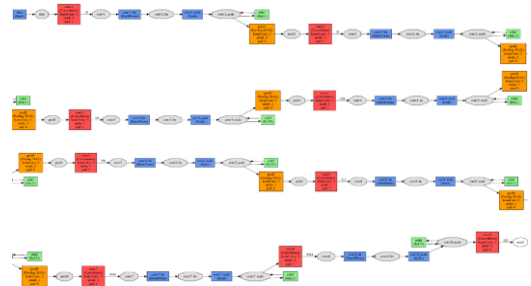
### Steps (still buggy)

- ▶ go to project directory
  - ➔ `$ cd work/codes/caffe_v1-projects/yolo_v2`
- ▶ converting Darknet configuration .cfg to Caffe .ptotxt:
  - ➔ `$ python create_yolo_prototxt.py tiny-yolo.cfg yolo_tiny`
- ▶ convert Darknet weight .weights to Caffe model .caffemodel
  - ➔ `$ python create_yolo_caffemodel.py yolo_tiny_deploy.prototxt tiny-yolo.weights yolo_tiny.caffemodel`
- ▶ use training (pretrained) model
- ▶ running Yolo model
  - ➔ `$ python yolo_detect.py yolo_tiny_deploy.prototxt yolo_tiny.caffemodel images/dog.jpg`

refer to <https://github.com/tsingjinyun/caffe-yolov2>

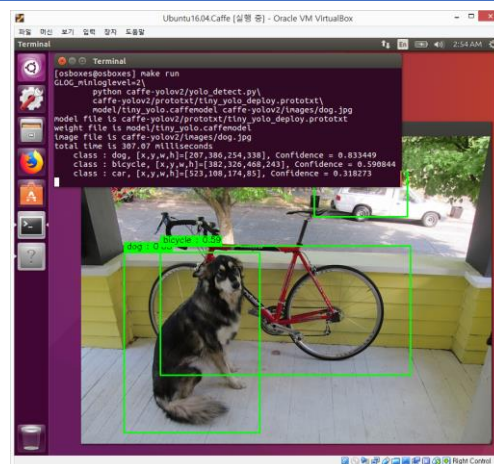
### Step in simple

- ▶ go to project directory
  - ➔ `$ cd work/codes/caffe_v1-projects/yolo_v2`
- ▶ get project
  - ➔ `$ make get`
- ▶ Run make
  - ➔ `$ make run`



31

## YOLO on Caffe



32



(주)퓨처디자인시스템

34051 대전광역시 유성구 문지로 193, KAIST 문지캠퍼스, F723호  
(042) 864-0211~0212 / [contact@future-ds.com](mailto:contact@future-ds.com) / [www.future-ds.com](http://www.future-ds.com)

Future Design Systems, Inc.

Faculty Wing F723, KAIST Munji Campus, 193 Munji-ro, Yuseong-gu, Daejeon 34051, Korea  
+82-042-864-0211~0212 / [contact@future-ds.com](mailto:contact@future-ds.com) / [www.future-ds.com](http://www.future-ds.com)



**FUTURE**  
Design Systems