Caffe V1 Examples

- LeNet and YOLO -

2019 – 2020

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

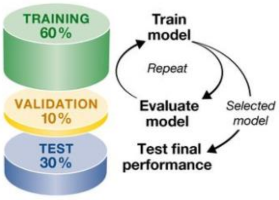
LeNet Example

- 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

2

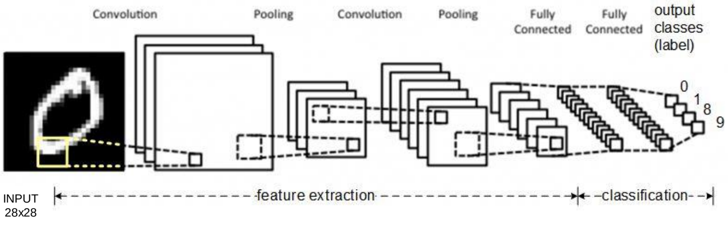
LeNet-5 for MNIST

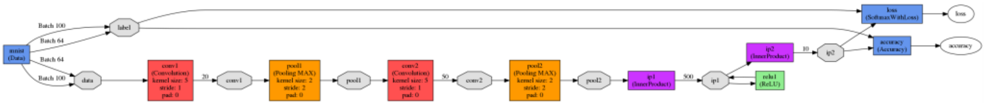
- MNIST dataset
  - Modified National Institute of Standards and Technology
  - Handwritten digits database
    - 10 classes: 0, 1, …, 9
    - training set: 60,000 training image
    - test set: 10,000 testing image



LeNet-5 for MNIST

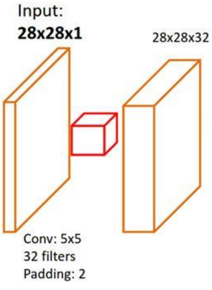
- LeNet is one of the popular convolutional networks, and works well on digit classification tasks.
  - 784 (28x28) inputs of black and white ➔ converted to floating number 0.0 ~ 1.0
  - 10 outputs representing digit 0 to 9



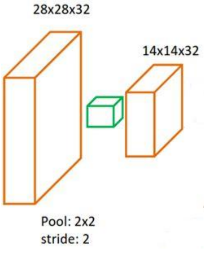


LeNet-5 for MNIST: layer

- 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

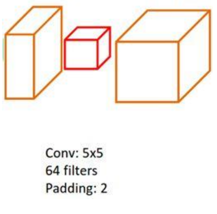


- 1^{st} pooling layer (sub-sampling)
  - Input: 32 features with 28x28
  - Max pooling filter: 5x5 (2x2 ?)
  - Convolution: stride 2
    - It generates ½ number of elements
  - Results in: 32 features of 14x14

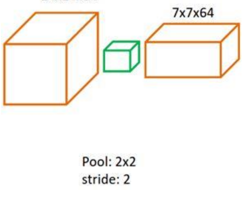


LeNet-5 for MNIST: layer

- 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

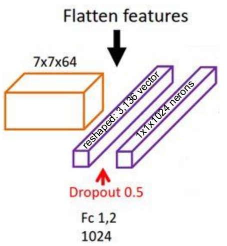


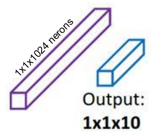
- 2^{nd} pooling
  - Input: 64 features with 14x14
  - Max pooling filter: 2x2
  - Convolution: stride 2
    - It generates ½ number of elements
  - Results in: 64 features of 7x7



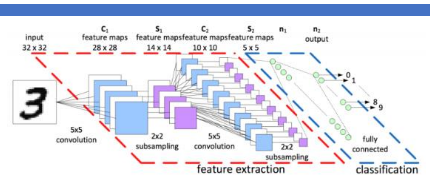
LeNet-5 for MNIST: layer

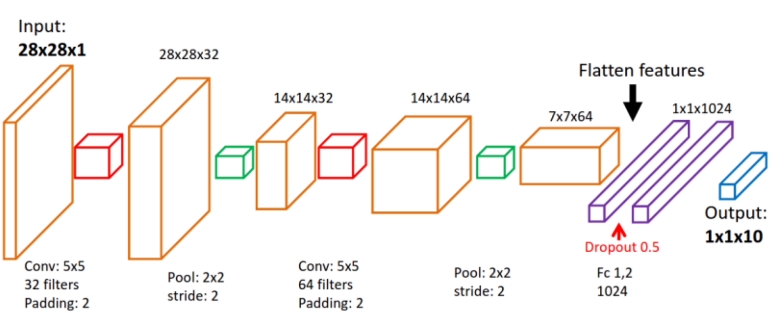
- fully connected layer
  - Input: 64 features with 7x7
  - Reshaping: 3-D array to 1-D vector
    - 64x7x7 → 3,136
  - Neurons: 1024
- read-out layer
  - Input: 1024 neurons
  - Output: 10 classes

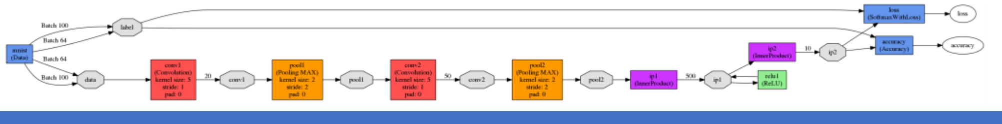




LeNet-5 for MNIST: all together

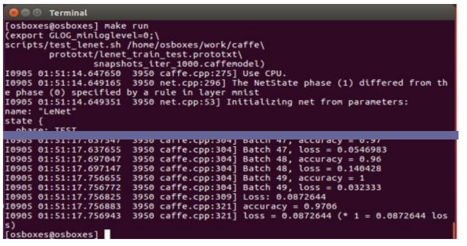






LeNet-5 for MNIST: running

- 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



9

LeNet-5 for MNIST: solver

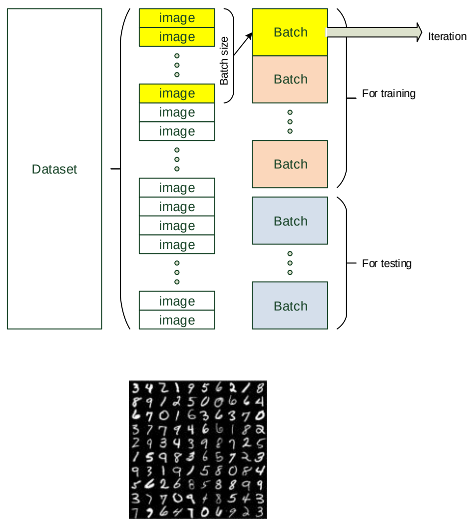
- net: network mode
- test_iter: iterations to test
- test_interval: interval between test
- base_lr: Learning Rate initial value
- display: iterations to show progress
- max_iter: max iterations for training.
- snapshot: iterations to store snapshot.
- solver_mode: CPU or GPU

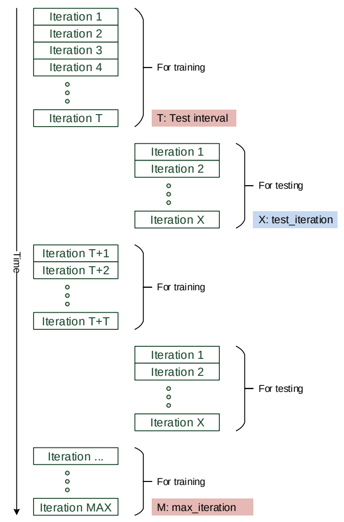
```
# MNIST lenet_solver.prototxt
net: "lenet_train_test.prototxt"
test_iter: 100
test_interval: 500
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"
gamma: 0.0001
power: 0.75
display: 100
max_iter: 10000
snapshot: 5000
snapshot_prefix: “snapshots"
solver_mode: CPU
```

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

10

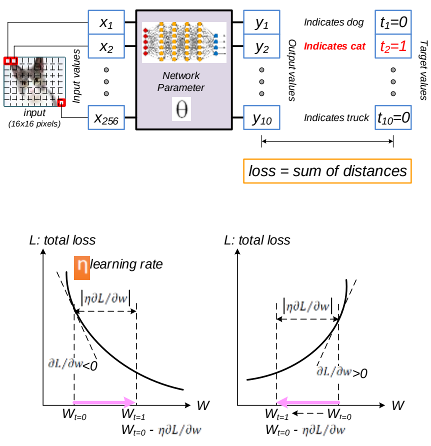
LeNet-5 for MNIST: solver

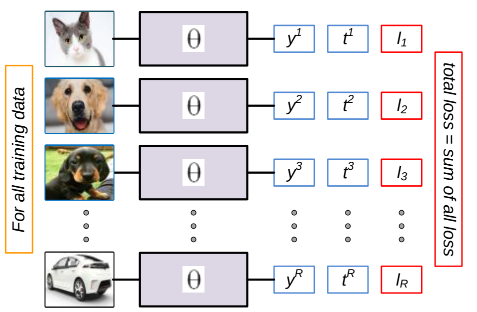




11

LeNet-5 for MNIST: solver





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

Sum of losses for R test images

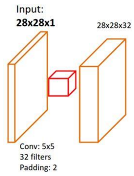
12

LeNet-5 for MNIST: net

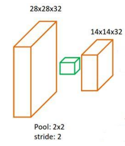
```
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "mnist"
  type: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

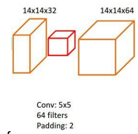
lenet_train_test.prototxt

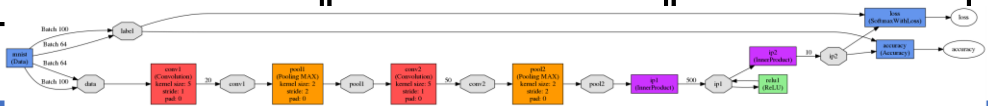
```
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: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

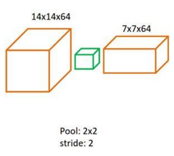


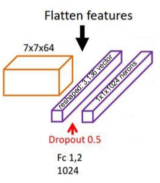




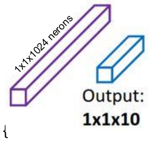
LeNet-5 for MNIST: net

```
    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"
  }
```

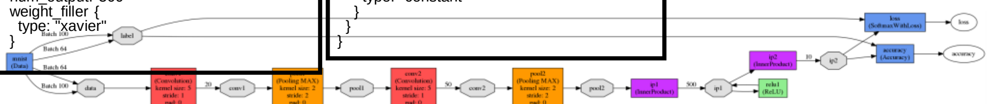





```
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10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

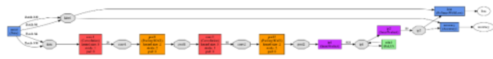


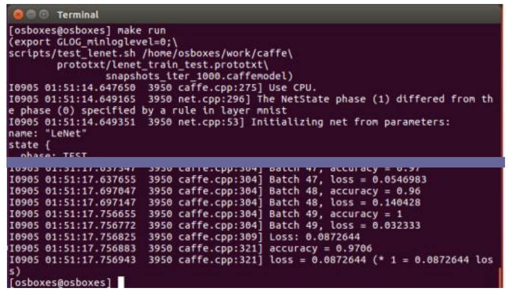

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```



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

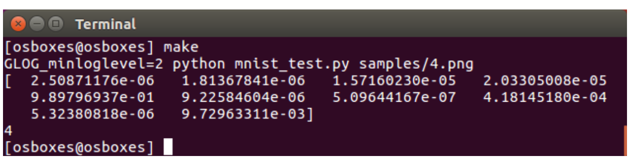


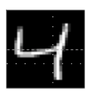


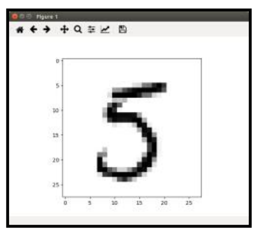
use 'display' for Ubuntu, 'fim' for Raspbian' to display image.

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"
}
```

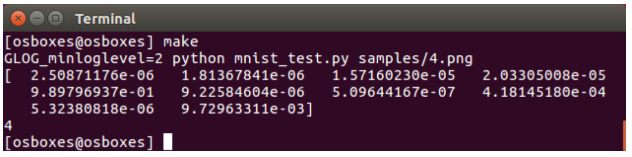
Remove

Remove

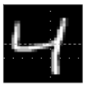
Add

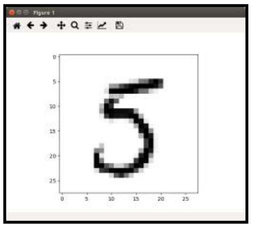
Run inference with sample image (another way)

- 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.





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

YOLO

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

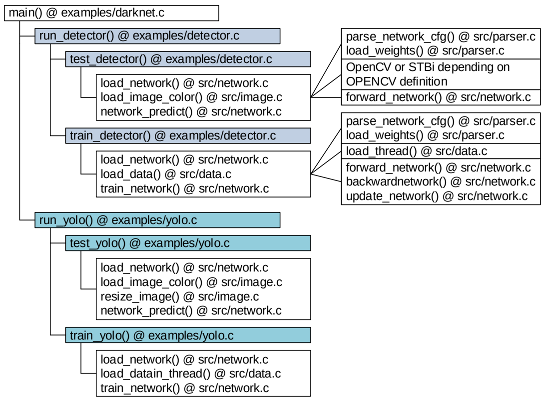
Darknet: The framework for YOLO

- 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.

Darknet example for detector

- Have a look at ‘darknet/examples/darknet.c’ file



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]
  - [avg] or [avgpool]
  - [dropout]
  - [lrn] 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
```

https://github.com/cvjena/darknet/blob/master/cfg/yolo.cfg

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 correponding factor once the number of steps is reached
  - max_batches: max number of "iterations"
  - i_snapshot_iteration: snapshow 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]

tiny-yolo.cfg

```
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   13 conv    512  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x 512
   14 conv    425  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 425
   15 detection
```

```
[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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=1

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

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

[convolutional]
size=1
stride=1
pad=1
filters=425
activation=linear

[region]
anchors = 0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828
bias_match=1
classes=80
coords=4
num=5
softmax=1
jitter=.2
rescore=0

object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1

absolute=1
thresh = .6
random=1
```

26

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   ->   416 x 416 x  16
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   13 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024
   14 conv    125  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 125
   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=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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=2

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

[maxpool]
size=2
stride=1

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

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

[convolutional]
size=1
stride=1
pad=1
filters=125
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
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

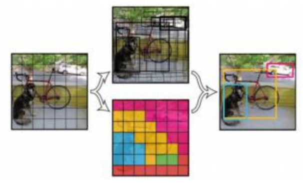
- YOLO (You Only Look Once)
  - An implementation of object-detection using Darknet.
  - YOLO: https://pjreddie.com/darknet/yolo/ ➔ YOLO3
  - YOLO (darknet): https://pjreddie.com/darknet/yolov1/
  - YOLOv2 (darknet): https://pjreddie.com/darknet/yolo/
  - YOLO (caffe): https://github.com/xingwangsfu/caffe-yolo
  - YOLO (TensorFlow: Train+Test): https://github.com/thtrieu/darkflow
  - YOLO (TensorFlow: Test): https://github.com/gliese581gg/YOLO_tensorflow

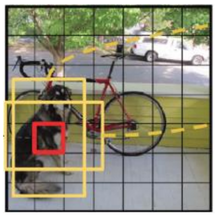
- Why it is called YOLO
  - A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.

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

YOLO model

- YOLO model





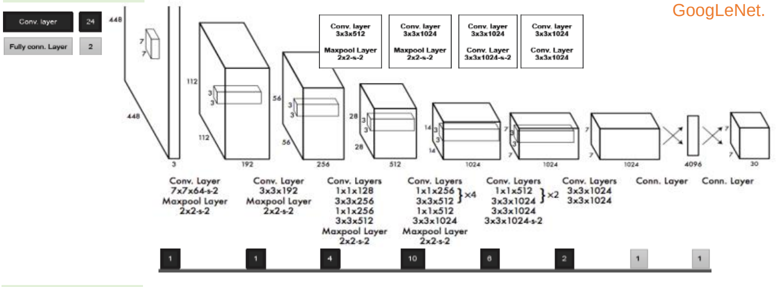
- divides the input image into a SxS grid. E.g., 7x7 grid
  - predict B bounding boxes. E.g., 2
  - each bounding box consists of 5 parameters (x, y, w, h, confidence).
  - (x, y) coordinates represent the center of the box
  - (w, h)

29

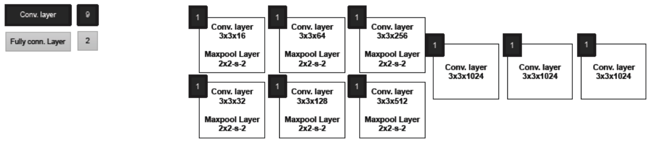
YOLO network

Inspired by GoogLeNet.

Normal YOLO



Tiny YOLO



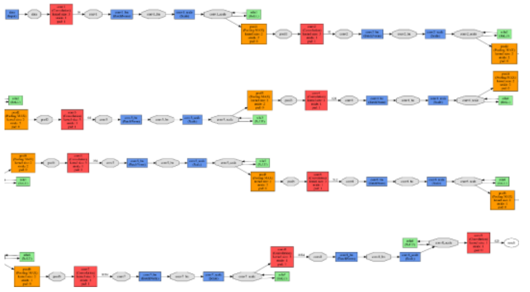
30

YOLO on Caffe

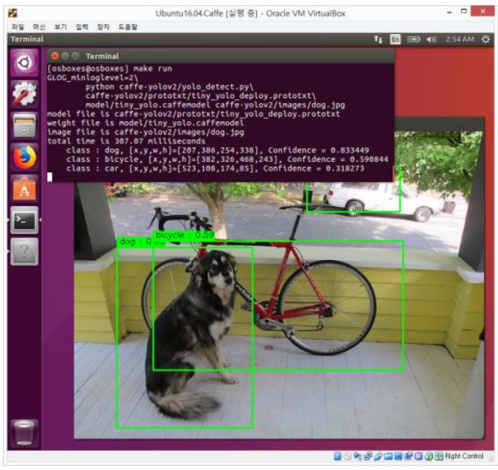
- Steps (still buggy)
  - go to project directory
    - $ cd work/codes/caffe_v1-projects/yolo_v2
  - converting Darknet configuration .cfg to Caffe .ptototxt:
    - $ python create_yolo_prototxt.py tiny-yolo.cfg yolo_tiny
  - convert Darknet weight .weights to Caffe model .caffemodel
    - $ python 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



YOLO on Caffe



㈜퓨쳐디자인시스템
34051 대전광역시 유성구 문지로 193, KAIST 문지캠퍼스, F723호
(042) 864-0211~0212 / contact@future-ds.com / 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 / www.future-ds.com

FUTURE
Design Systems