目录

[YOLOV4框架使用说明 2](#_Toc41998351)

[一、 windows编译 2](#_Toc41998352)

[二、 linux 编译 2](#_Toc41998353)

[三、 训练参数和cfg-files解读 3](#_Toc41998354)

[3.1. 目标检测训练基础参数配置： 3](#_Toc41998355)

[3.2. CFG-Parameters in the [net] section: 4](#_Toc41998356)

[3.3. CFG-Parameters in the different layers 9](#_Toc41998357)

[3.4. Use Yolo v3 in other frameworks 16](#_Toc41998358)

[3.5. 多GPU训练参数 17](#_Toc41998359)

[四、 训练用户检测器 18](#_Toc41998360)

[五、 停止训练条件 25](#_Toc41998361)

[六、 目标检测精度的提高 29](#_Toc41998362)

[七、 DLL和SO库封装 34](#_Toc41998363)

[八、 Command line 使用 37](#_Toc41998364)

[九、 指标解析 39](#_Toc41998365)

[十、 其他指令 40](#_Toc41998366)

# YOLOV4框架使用说明

## windows编译

配置要求： Visual Studio 2015/2017/2019, CUDA > 10.0, cuDNN > 7.0, and OpenCV > 2.4

1. 打开 yourpath/darknet-master/build/darknet/ 目录，使用vs15/vs17打开所需要的工程即可，按照工程属性，配置cuda，opencv，cudnn等的include path 和lib path，然后选择合适的版本编译即可。
2. 自行编译源码：将所有的.c ;.cu;和相应的头文件添加到创建的vs工程中，然后添加依赖的include，lib等相应文件，执行编译即可。

## linux 编译

配置要求：CUDA > 10.0, cuDNN > 7.0, and OpenCV > 2.4

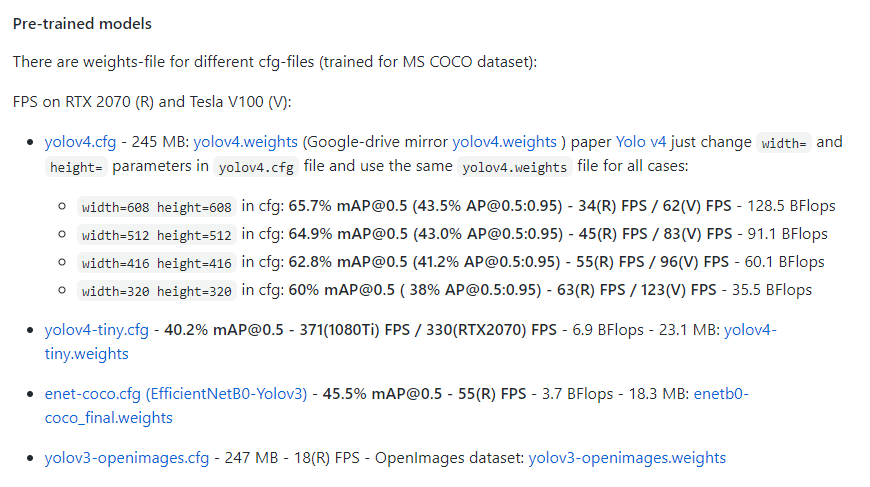
打开 yourpath/darknet-master 进入主目录，使用make进行编译

相关编译参数如下：

* GPU=1 to build with CUDA to accelerate by using GPU (CUDA should be in /usr/local/cuda)
* CUDNN=1 to build with cuDNN v5-v7 to accelerate training by using GPU (cuDNN should be in /usr/local/cudnn)
* CUDNN\_HALF=1 to build for Tensor Cores (on Titan V / Tesla V100 / DGX-2 and later) speedup Detection 3x, Training 2x
* OPENCV=1 to build with OpenCV 4.x/3.x/2.4.x - allows to detect on video files and video streams from network cameras or web-cams
* DEBUG=1 to bould debug version of Yolo
* OPENMP=1 to build with OpenMP support to accelerate Yolo by using multi-core CPU
* LIBSO=1 to build a library darknet.so and binary runable file uselib that uses this library. Or you can try to run so LD\_LIBRARY\_PATH=./:$LD\_LIBRARY\_PATH ./uselib test.mp4 How to use this SO-library from your own code - you can look at C++ example: <https://github.com/AlexeyAB/darknet/blob/master/src/yolo_console_dll.cpp> or use in such a way: LD\_LIBRARY\_PATH=./:$LD\_LIBRARY\_PATH ./uselib data/coco.names cfg/yolov4.cfg yolov4.weights test.mp4
* ZED\_CAMERA=1 to build a library with ZED-3D-camera support (should be ZED SDK installed), then run LD\_LIBRARY\_PATH=./:$LD\_LIBRARY\_PATH ./uselib data/coco.names cfg/yolov4.cfg yolov4.weights zed\_camera

修改好自己所需要的配置参数；执行make -j32 即可编译

## 训练参数和cfg-files解读



### 目标检测训练基础参数配置：

(**change width=512 height=512 in cfg-file**)

mini\_batch\_size = batch / subdivisions, **so higher subdivisions= - lower accuracy (AP):**

* for 32 GB GPU-VRAM set subdivisions=8 in cfg-file ([yolov4.weights](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights) is trained on V100-32GB)
* for 16-24 GPU-VRAM set subdivisions=16 in cfg-file
* for 8-12 GB GPU-VRAM set subdivisions=32 in cfg-file (if Out Of Memory occurs - set random=1.34 for [yolo] layers)
* if you compiled Darknet without OpenCV and disabled mosaic=0 then accuracy may be ~2% lower

### CFG-Parameters in the [net] section:

1. [net] section
   * batch=1 - number of samples (images, letters, ...) which will be precossed in one batch
   * subdivisions=1 - number of mini\_batches in one batch, size mini\_batch = batch/subdivisions, so GPU processes mini\_batch samples at once, and the weights will be updated for batch samples (1 iteration processes batch images)
   * width=416 - network size (width), so every image will be resized to the network size during Training and Detection
   * height=416 - network size (height), so every image will be resized to the network size during Training and Detection
   * channels=3 - network size (channels), so every image will be converted to this number of channels during Training and Detection
   * inputs=256 - network size (inputs) is used for non-image data: letters, prices, any custom data

**For training only**

* Data augmentation:
  + angle=0 - randomly rotates images during training (classification only)
  + saturation = 1.5 - randomly changes saturation of images during training
  + exposure = 1.5 - randomly changes exposure (brightness) during training
  + hue=.1 - randomly changes hue (color) during training <https://en.wikipedia.org/wiki/HSL_and_HSV>
  + blur=1 - blur will be applied randomly in 50% of the time: if 1 - will be blured background except objects with blur\_kernel=31, if >1 - will be blured whole image with blur\_kernel=blur (only for detection and if OpenCV is used)
  + min\_crop=224 - minimum size of randomly cropped image (classification only)
  + max\_crop=448 - maximum size of randomly cropped image (classification only)
  + aspect=.75 - aspect ration can be changed during croping from 0.75 - to 1/0.75 (classification only)
  + letter\_box=1 - keeps aspect ratio of loaded images during training (detection training only, but to use it during detection-inference - use flag -letter\_box at the end of detection command)
  + data augmentation in the last [yolo]-layer
    - jitter=0.3 - randomly changes size of image and its aspect ratio from x(1 - 2\*jitter) to x(1 + 2\*jitter)
    - random=1 - randomly resizes network size after each 10 batches (iterations) from /1.4 to x1.4 with keeping initial aspect ratio of network size
* Optimizator:
  + momentum=0.9 - accumulation of movement, how much the history affects the further change of weights (optimizer)
  + decay=0.0005 - a weaker updating of the weights for typical features, it eliminates dysbalance in dataset (optimizer) <http://cs231n.github.io/neural-networks-3/>
  + learning\_rate=0.001 - initial learning rate for training
  + burn\_in=1000 - initial burn\_in will be processed for the first 1000 iterations, current\_learning rate = learning\_rate \* pow(iterations / burn\_in, power) = 0.001 \* pow(iterations/1000, 4) where is power=4 by default
  + max\_batches = 500200 - the training will be processed for this number of iterations (batches)
  + policy=steps - policy for changing learning rate: constant (by default), sgdr, steps, step, sig, exp, poly, random (f.e., if policy=random - then current learning rate will be changed in this way = learning\_rate \* pow(rand\_uniform(0,1), power))
  + power=4 - if policy=poly - the learning rate will be = learning\_rate \* pow(1 - current\_iteration / max\_batches, power)
  + sgdr\_cycle=1000 - if policy=sgdr - the initial number of iterations in cosine-cycle
  + sgdr\_mult=2 - if policy=sgdr - multiplier for cosine-cycle <https://towardsdatascience.com/https-medium-com-reina-wang-tw-stochastic-gradient-descent-with-restarts-5f511975163>
  + steps=8000,9000,12000 - if policy=steps - at these numbers of iterations the learning rate will be multiplied by scales factor
  + scales=.1,.1,.1 - if policy=steps - f.e. if steps=8000,9000,12000, scales=.1,.1,.1 and the current iteration number is 10000 then current\_learning\_rate = learning\_rate \* scales[0] \* scales[1] = 0.001 \* 0.1 \* 0.1 = 0.00001

For training Recurrent networks:

* Object Detection/Tracking on Video - if [conv-lstm] or [crnn] layers are used in additional to [connected] and [convolutional] layers
* Text generation - if [lstm] or [rnn] layers are used in additional to [connected] layers
  + track=1 - if is set 1 then the training will be performed in Recurrents-tyle for image sequences
  + time\_steps=16 - training will be performed for a random image sequence that contains 16 images from train.txt file
    - for [convolutional]-layers: mini\_batch = time\_steps\*batch/subdivisions
    - for [conv\_lstm]-recurrent-layers: mini\_batch = batch/subdivisions and sequence=16
  + augment\_speed=3 - if set 3 then can be used each 1st, 2nd or 3rd image randomly, i.e. can be used 16 images with indexes 0, 1, 2, ... 15 or 110, 113, 116, ... 155 from train.txt file
  + sequential\_subdivisions=8 - lower value increases the sequence of images, so if time\_steps=16 batch=16 sequential\_subdivisions=8, then will be loaded time\_steps\*batch/sequential\_subdivisions = 16\*16/8 = 32 sequential images with the same data-augmentation, so the model will be trained for sequence of 32 video-frames
  + seq\_scales=0.5, 0.5 - increasing sequence of images at some steps, i.e. the coefficients to which the original sequential\_subdivisions value will be multiplied (and batch will be dividied, so the weights will be updated rarely) at correspond steps if is used policy=steps or policy=sgdr

### CFG-Parameters in the different layers

**Image processing [N x C x H x W]:**

* [convolutional] - convolutional layer
  + batch\_normalize=1 - if 1 - will be used batch-normalization, if 0 will not (0 by default)
  + filters=64 - number of kernel-filters (1 by default)
  + size=3 - kernel\_size of filter (1 by default)
  + groups = 32 - number of groups for grouped-convolutional (depth-wise) (1 by default)
  + stride=1 - stride (offset step) of kernel filter (1 by default)
  + padding=1 - size of padding (0 by default)
  + pad=1 - if 1 will be used padding = size/2, if 0 the will be used parameter padding= (0 by default)
  + dilation=1 - size of dilation (1 by default)
  + activation=leaky - activation function after convolution: logistic (by default), loggy, relu, elu, selu, relie, plse, hardtan, lhtan, linear, ramp, leaky, tanh, stair
* [activation] - separate activation layer
  + activation=leaky - activation function: linear (by default), loggy, relu, elu, selu, relie, plse, hardtan, lhtan, linear, ramp, leaky, tanh, stair
* [batchnorm] - separate Batch-normalization layer
* [maxpool] - max-pooling layer (the maximum value)
  + size=2 - size of max-pooling kernel
  + stride=2 - stirde (offset step) of max-pooling kernel
* [avgpool] - average pooling layer input W x H x C -> output 1 x 1 x C
* [shortcut] - residual connection (ResNet)
  + from=-3 - relative layer number, preforms element-wise adding of two layers: previous-layer and layer specified in from= parameter
  + activation=linear - activation function after shortcut/residual connection (linear by default)
* [upsample] - upsample layer (increase W x H resolution of input by duplicating elements)
  + stride=2 - factor for increasing both Width and Height (new\_w = w\*stride, new\_h = h\*stride)
* [scale\_channels] - scales channels (squeeze-and-excitation blocks)
  + from=-3 - relative layer number, performs multiplication of all elements of channel N from layer -3, by one element of channel N from the previous layer -1 (i.e. for(int i=0; i < b\*c\*h\*w; ++i) output[i] = from\_layer[i] \* previous\_layer[i/(w\*h)]; )
  + activation=linear - activation function after scale\_channels-layer (linear by default)
* [reorg3d] - reorg layer (resize W x H x C)
  + stride=2 - if reverse=0 input will be resized to W/2 x H/2 x C4*, if*reverse=1*then*W2 x H\*2 x C/4`, (1 by default)
  + reverse=1 - if 0(by default) then decrease WxH, if 1 then increase WxH (0 by default)
* [reorg] - OLD reorg layer from Yolo v2 - has incorrect logic (resize W x H x C) - depracated
  + stride=2 - if reverse=0 input will be resized to W/2 x H/2 x C4*, if*reverse=1 *then* W2 x H\*2 x C/4`, (1 by default)
  + reverse=1 - if 0(by default) then decrease WxH, if 1 then increase WxH (0 by default)
* [route] - concatenation layer, Concat for several input-layers, or Identity for one input-layer
  + layers = -1, 61 - layers that will be concatenated, output: W x H x C\_layer\_1 + C\_layer\_2
    - if index < 0, then it is relative layer number (-1 means previous layer)
    - if index >= 0, then it is absolute layer number
* [yolo] - detection layer for Yolo v3
  + mask = 3,4,5 - indexes of anchors which are used in this [yolo]-layer
  + anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326 - initial sizes if bounded\_boxes that will be adjusted
  + num=9 - total number of anchors
  + classes=80 - number of classes of objects which can be detected
  + ignore\_thresh = .7 - keeps duplicated detections if IoU(detect, truth) > ignore\_thresh, which will be fused during NMS (is used for training only)
  + truth\_thresh = 1 - adjusts duplicated detections if IoU(detect, truth) > truth\_thresh, which will be fused during NMS (is used for training only)
  + jitter=.3 - randomly crops and resizes images with changing aspect ratio from x(1 - 2\*jitter) to x(1 + 2\*jitter) (data augmentation parameter is used only from the last layer)
  + random=1 - randomly resizes network for each 10 iterations from 1/1.4 to 1.4(data augmentation parameter is used only from the last layer)
* [crnn] - convolutional RNN-layer (recurrent)
  + batch\_normalize=1 - if 1 - will be used batch-normalization, if 0 will not (0 by default)
  + size=1 - convolutional kernel\_size of filter (1 by default)
  + pad=0 - if 1 will be used padding = size/2, if 0 the will be used parameter padding= (0 by default)
  + output = 1024 - number of kernel-filters in one output convolutional layer (1 by default)
  + hidden=1024 - number of kernel-filters in two (input and hidden) convolutional layers (1 by default)
  + activation=leaky - activation function for each of 3 convolutional-layers in the [crnn]-layer (logistic by default)
* [conv\_lstm] - convolutional LSTM-layer (recurrent)
  + batch\_normalize=1 - if 1 - will be used batch-normalization, if 0 will not (0 by default)
  + size=3 - convolutional kernel\_size of filter (1 by default)
  + padding=1 - convolutional size of padding (0 by default)
  + pad=1 - if 1 will be used padding = size/2, if 0 the will be used parameter padding= (by default)
  + stride=1 - convolutional stride (offset step) of kernel filter (1 by default)
  + dilation=1 - convolutional size of dilation (1 by default)
  + output=256 - number of kernel-filters in each of 8 or 11 convolutional layers (1 by default)
  + groups=4 - number of groups for grouped-convolutional (depth-wise) (1 by default)
  + state\_constrain=512 - constrains LSTM-state values [-512; +512] after each inference (time\_steps\*32 by default)
  + peephole=0 - if 1 then will be used Peephole (additional 3 conv-layers), if 0 will not (1 by default)
  + activation=leaky - activation function for each of 8 or 11 convolutional-layers in the [conv\_lstm]-layer (linear by default)

**Free-form data processing [Inputs]:**

* [connected] - fully connected layer
  + output=256 - number of outputs (1 by default), so number of connections is equal to inputs\*outputs
  + activation=leaky - activation after layer (logistic by default)
* [dropout] - dropout layer
  + probability=0.5 - dropout probability - what part of inputs will be zeroed (0.5 = 50% by default)
* [softmax] - SoftMax CE (cross entropy) layer - Categorical cross-entropy for multi-class classification
* [cost] - cost layer calculates (linear)Delta and (squared)Loss
  + type=sse - cost type: sse (L2), masked, smooth (smooth-L1) (SSE by default)
* [rnn] - fully connected RNN-layer (recurrent)
  + batch\_normalize=1 - if 1 - will be used batch-normalization, if 0 will not (0 by default)
  + output = 1024 - number of outputs in one connected layer (1 by default)
  + hidden=1024 - number of outputs in two (input and hidden) connected layers (1 by default)
  + activation=leaky - activation after layer (logistic by default)
* [lstm] - fully connected LSTM-layer (recurrent)
  + batch\_normalize=1 - if 1 - will be used batch-normalization, if 0 will not (0 by default)
  + output = 1024 - number of outputs in all connected layers (1 by default)
* [gru] - fully connected GRU-layer (recurrent)
  + batch\_normalize=1 - if 1 - will be used batch-normalization, if 0 will not (0 by default)
  + output = 1024 - number of outputs in all connected layers (1 by default)

### Use Yolo v3 in other frameworks

**CLICK ME** - Use Yolo in other frameworks

* **OpenCV-dnn** the fastest implementation for CPU (x86/ARM-Android), OpenCV can be compiled with [OpenVINO-backend](https://github.com/opencv/opencv/wiki/Intel's-Deep-Learning-Inference-Engine-backend) for running on (Myriad X / USB Neural Compute Stick / Arria FPGA), use yolov3.weights/cfg with: [C++ example](https://github.com/opencv/opencv/blob/8c25a8eb7b10fb50cda323ee6bec68aa1a9ce43c/samples/dnn/object_detection.cpp#L192-L221) or [Python example](https://github.com/opencv/opencv/blob/8c25a8eb7b10fb50cda323ee6bec68aa1a9ce43c/samples/dnn/object_detection.py#L129-L150)
* Converting Yolo v3 models to TensorFlow and OpenVINO(IR) models: <https://github.com/AlexeyAB/darknet/wiki/Converting-Yolo-v3-models-to-TensorFlow-and-OpenVINO(IR)-models>
  + **TensorFlow:** convert yolov3.weights/cfg files to yolov3.ckpt/pb/meta: by using [mystic123](https://github.com/mystic123/tensorflow-yolo-v3) or [jinyu121](https://github.com/jinyu121/DW2TF) projects, and [TensorFlow-lite](https://www.tensorflow.org/lite/guide/get_started#2_convert_the_model_format)
  + **Intel OpenVINO 2019 R1:** (Myriad X / USB Neural Compute Stick / Arria FPGA): read this [manual](https://software.intel.com/en-us/articles/OpenVINO-Using-TensorFlow#converting-a-darknet-yolo-model)
* **PyTorch > ONNX > CoreML > iOS** how to convert cfg/weights-files to pt-file: [ultralytics/yolov3](https://github.com/ultralytics/yolov3#darknet-conversion) and [iOS App](https://itunes.apple.com/app/id1452689527)
* **TensorRT** for YOLOv3 (-70% faster inference): [Yolo is natively supported in DeepStream 4.0](https://news.developer.nvidia.com/deepstream-sdk-4-now-available/) read [PDF](https://docs.nvidia.com/metropolis/deepstream/Custom_YOLO_Model_in_the_DeepStream_YOLO_App.pdf)21
* **TVM** - compilation of deep learning models (Keras, MXNet, PyTorch, Tensorflow, CoreML, DarkNet) into minimum deployable modules on diverse hardware backends (CPUs, GPUs, FPGA, and specialized accelerators): <https://tvm.ai/about>
* **OpenDataCam** - It detects, tracks and counts moving objects by using Yolo: <https://github.com/opendatacam/opendatacam#-hardware-pre-requisite>
* **Netron** - Visualizer for neural networks: <https://github.com/lutzroeder/netron>

### 多GPU训练参数&&继续训练

1. Train it first on 1 GPU for like 1000 iterations: darknet.exe detector train cfg/coco.data cfg/yolov4.cfg yolov4.conv.137
2. **Then stop and by using partially-trained** model /backup/yolov4\_1000.weights run training with multigpu (up to 4 GPUs): darknet.exe detector train cfg/coco.data cfg/yolov4.cfg /backup/yolov4\_1000.weights -gpus 0,1,2,3

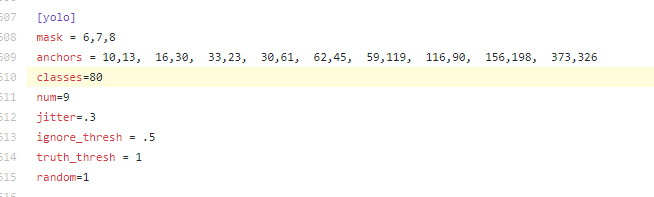
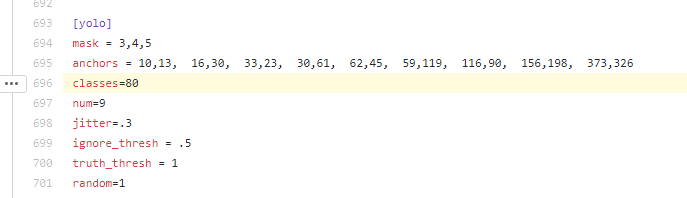
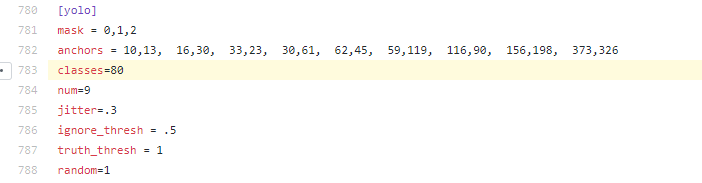
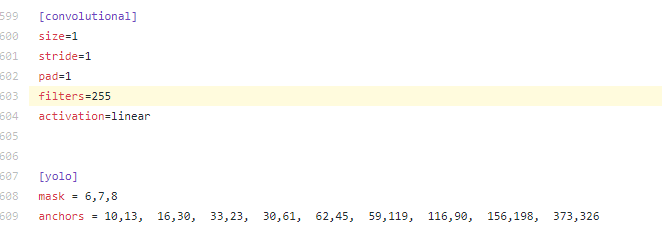
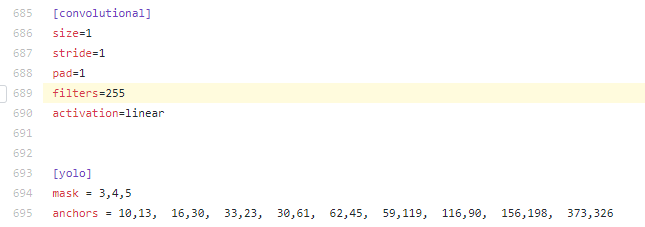
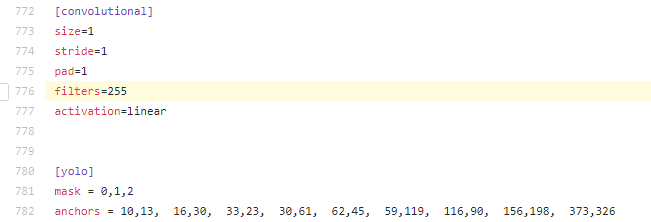
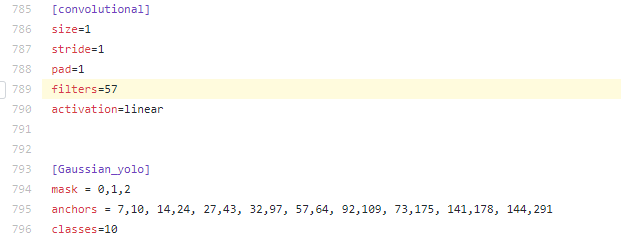
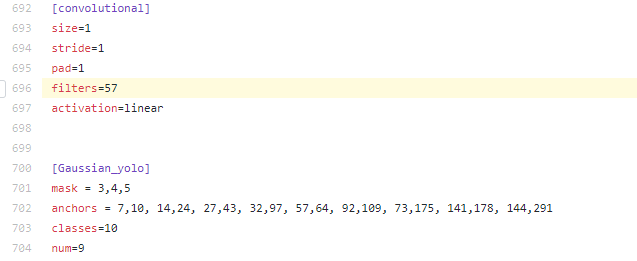
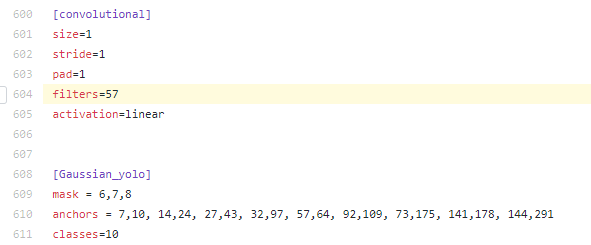
If you get a Nan, then for some datasets better to decrease learning rate, for 4 GPUs set **learning\_rate = 0,00065**(i.e. learning\_rate = 0.00261 / GPUs). In this case also increase 4x times burn\_in = in your cfg-file. I.e. use burn\_in = 4000 instead of 1000.

## 训练用户检测器

### Training Yolo v4 (and v3)

1. For training cfg/yolov4-custom.cfg download the pre-trained weights-file (162 MB): [yolov4.conv.137](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137) (Google drive mirror [yolov4.conv.137](https://drive.google.com/open?id=1JKF-bdIklxOOVy-2Cr5qdvjgGpmGfcbp) )
2. Create file yolo-obj.cfg with the same content as in yolov4-custom.cfg (or copy yolov4-custom.cfg to yolo-obj.cfg)

and:

* change line batch to [batch=64](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L3)
* change line subdivisions to [subdivisions=16](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L4)（depend your gpu memory）
* change line max\_batches to (classes\*2000 but not less than number of training images, and not less than 6000), f.e. [max\_batches=6000](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L20) if you train for 3 classes
* change line steps to 80% and 90% of max\_batches, f.e. [steps=4800,5400](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L22)
* set network size width=416 height=416 or any value multiple of 32: 
* change line classes=80 to your number of objects in each of 3 [yolo]-layers: （3个地方）  
* change [filters=255] to filters=(classes + 5)x3 in the 3 [convolutional] before each [yolo] layer, keep in mind that it only has to be the last [convolutional] before each of the [yolo] layers. （3个地方）  
* when using [[Gaussian\_yolo]](https://github.com/AlexeyAB/darknet/blob/6e5bdf1282ad6b06ed0e962c3f5be67cf63d96dc/cfg/Gaussian_yolov3_BDD.cfg#L608) layers, change [filters=57] filters=(classes + 9)x3 in the 3 [convolutional] before each [Gaussian\_yolo] layer（gaussian\_yolov3\_BDD.cfg）

So if classes=1 then should be filters=18. If classes=2 then write filters=21.

**(Do not write in the cfg-file: filters=(classes + 5)x3)**

(Generally filters depends on the classes, coords and number of masks, i.e. filters=(classes + coords + 1)\*<number of mask>, where mask is indices of anchors. If mask is absence, then filters=(classes + coords + 1)\*num)

So for example, for 2 objects, your file yolo-obj.cfg should differ from yolov4-custom.cfg in such lines in each of **3** [yolo]-layers:

[convolutional]

filters=21

[region]

classes=2

1. Create file obj.names in the directory build\darknet\x64\data\, with objects names - each in new line
2. Create file obj.data in the directory build\darknet\x64\data\, containing (where **classes = number of objects**):

classes= 2

train = data/train.txt

valid = data/test.txt

names = data/obj.names

backup = backup/

1. Put image-files (.jpg/.png/.bmp) of your objects in the directory build\darknet\x64\data\obj\
2. You should label each object on images from your dataset. Use this visual GUI-software  (<https://github.com/AlexeyAB/Yolo_mark>)for marking bounded boxes of objects and generating annotation files for Yolo v2 & v3: or use labelimage ;详情参照标定软件。
3. Create file train.txt in directory build\darknet\x64\data\, with filenames of your images, each filename in new line, with path relative to darknet.exe, for example containing:

data/obj/img1.jpg

data/obj/img2.jpg

data/obj/img3.jpg

1. Download pre-trained weights for the convolutional layers and put to the directory build\darknet\x64

* for yolov4.cfg, yolov4-custom.cfg (162 MB): [yolov4.conv.137](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137) (Google drive mirror [yolov4.conv.137](https://drive.google.com/open?id=1JKF-bdIklxOOVy-2Cr5qdvjgGpmGfcbp) )
* for csresnext50-panet-spp.cfg (133 MB): [csresnext50-panet-spp.conv.112](https://drive.google.com/file/d/16yMYCLQTY_oDlCIZPfn_sab6KD3zgzGq/view?usp=sharing)
* for yolov3.cfg, yolov3-spp.cfg (154 MB): [darknet53.conv.74](https://pjreddie.com/media/files/darknet53.conv.74)
* for yolov3-tiny-prn.cfg , yolov3-tiny.cfg (6 MB): [yolov3-tiny.conv.11](https://drive.google.com/file/d/18v36esoXCh-PsOKwyP2GWrpYDptDY8Zf/view?usp=sharing)
* for enet-coco.cfg (EfficientNetB0-Yolov3) (14 MB): [enetb0-coco.conv.132](https://drive.google.com/file/d/1uhh3D6RSn0ekgmsaTcl-ZW53WBaUDo6j/view?usp=sharing)

1. **Start training** by using the command line: darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137

**当无法显示时，需要添加 -dont\_show**

To train on Linux use command: ./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137 (just use ./darknet instead of darknet.exe)

* (file yolo-obj\_last.weights will be saved to the build\darknet\x64\backup\ for each 100 iterations)
* (file yolo-obj\_xxxx.weights will be saved to the build\darknet\x64\backup\ for each 1000 iterations)
* (to disable Loss-Window use darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -dont\_show, if you train on computer without monitor like a cloud Amazon EC2)
* (to see the mAP & **Loss-chart during training on remote server without GUI**, use command**darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -dont\_show -mjpeg\_port 8090 -map** then open URL http://ip-address:8090 in Chrome/Firefox browser)
* For training with mAP (mean average precisions) calculation for each 4 Epochs (set valid=valid.txt or train.txt in obj.data file) and run: darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -map

1. After training is complete - get result yolo-obj\_final.weights from path build\darknet\x64\backup\

* After each 100 iterations you can stop and later start training from this point. For example, after 2000 iterations you can stop training, and later just start training using: darknet.exe detector train data/obj.data yolo-obj.cfg backup\yolo-obj\_2000.weights

(in the original repository <https://github.com/pjreddie/darknet> the weights-file is saved only once every 10 000 iterations if(iterations > 1000))

* Also you can get result earlier than all 45000 iterations.

**Note:** If during training you see nan values for avg (loss) field - then training goes wrong, but if nan is in some other lines - then training goes well.

**Note:** If you changed width= or height= in your cfg-file, then new width and height must be divisible by 32.

**Note:** After training use such command for detection: darknet.exe detector test data/obj.data yolo-obj.cfg yolo-obj\_8000.weights

**Note:** if error Out of memory occurs then in .cfg-file you should increase subdivisions=16, 32 or 64: [link](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L4)

## 停止训练条件

Usually sufficient 2000 iterations for each class(object), but not less than number of training images and not less than 6000 iterations in total.

But for a more precise definition when you should stop training, use the following manual:

1. During training, you will see varying indicators of error, and you should stop when no longer decreases **0.XXXXXXX avg**:

Region Avg IOU: 0.798363, Class: 0.893232, Obj: 0.700808, No Obj: 0.004567, Avg Recall: 1.000000, count: 8 Region Avg IOU: 0.800677, Class: 0.892181, Obj: 0.701590, No Obj: 0.004574, Avg Recall: 1.000000, count: 8

**9002**: 0.211667, **0.60730 avg**, 0.001000 rate, 3.868000 seconds, 576128 images Loaded: 0.000000 seconds

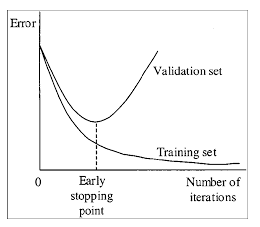
* **9002** - iteration number (number of batch)
* **0.60730 avg** - average loss (error) - **the lower, the better**

When you see that average loss **0.xxxxxx avg** no longer decreases at many iterations then you should stop training. The final avgerage loss can be from 0.05 (for a small model and easy dataset) to 3.0 (for a big model and a difficult dataset).

Or if you train with flag -map then you will see mAP indicator Last accuracy mAP@0.5 = 18.50% in the console - this indicator is better than Loss, so train while mAP increases.

1. Once training is stopped, you should take some of last .weights-files from darknet\build\darknet\x64\backup and choose the best of them:

For example, you stopped training after 9000 iterations, but the best result can give one of previous weights (7000, 8000, 9000). It can happen due to overfitting. **Overfitting** - is case when you can detect objects on images from training-dataset, but can't detect objects on any others images. You should get weights from **Early Stopping Point**:



To get weights from Early Stopping Point:

2.1. At first, in your file obj.data you must specify the path to the validation dataset valid = valid.txt (format of valid.txt as in train.txt), and if you haven't validation images, just copy data\train.txt to data\valid.txt.

2.2 If training is stopped after 9000 iterations, to validate some of previous weights use this commands:

(If you use another GitHub repository, then use darknet.exe detector recall... instead of darknet.exe detector map...)

* darknet.exe detector map data/obj.data yolo-obj.cfg backup\yolo-obj\_7000.weights
* darknet.exe detector map data/obj.data yolo-obj.cfg backup\yolo-obj\_8000.weights
* darknet.exe detector map data/obj.data yolo-obj.cfg backup\yolo-obj\_9000.weights

And comapre last output lines for each weights (7000, 8000, 9000):

Choose weights-file **with the highest mAP (mean average precision)** or IoU (intersect over union)

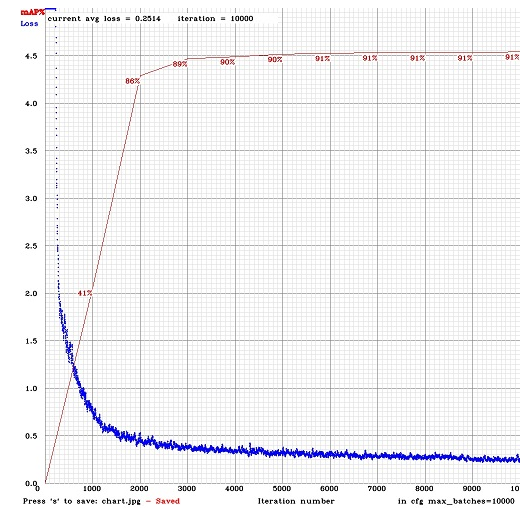
For example, **bigger mAP** gives weights yolo-obj\_8000.weights - then **use this weights for detection**.

Or just train **with -map flag:**

darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -map

So you will see mAP-chart (red-line) in the Loss-chart Window. mAP will be calculated for each 4 Epochs using valid=valid.txt file that is specified in obj.data file (1 Epoch = images\_in\_train\_txt / batch iterations)

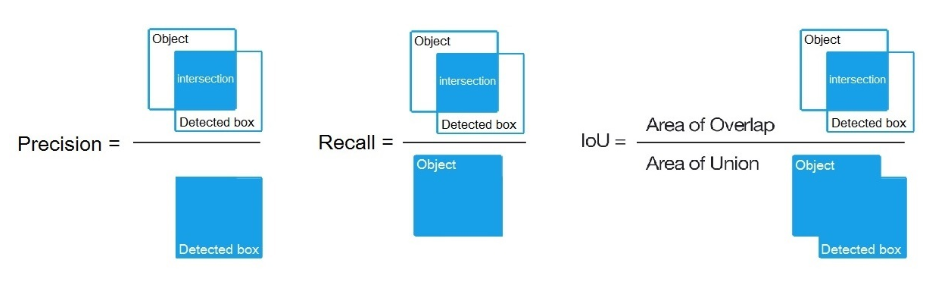
(to change the max x-axis value - change [max\_batches=](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L20) parameter to 2000\*classes, f.e. max\_batches=6000 for 3 classes)



Example of custom object detection: darknet.exe detector test data/obj.data yolo-obj.cfg yolo-obj\_8000.weights

* **IoU** (intersect over union) - average instersect over union of objects and detections for a certain threshold = 0.24
* **mAP** (mean average precision) - mean value of average precisions for each class, where average precision is average value of 11 points on PR-curve for each possible threshold (each probability of detection) for the same class (Precision-Recall in terms of PascalVOC, where Precision=TP/(TP+FP) and Recall=TP/(TP+FN) ), page-11: <http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf>

**mAP** is default metric of precision in the PascalVOC competition, **this is the same as AP50** metric in the MS COCO competition. In terms of Wiki, indicators Precision and Recall have a slightly different meaning than in the PascalVOC competition, but **IoU always has the same meaning**.



## 目标检测精度的提高

1. Before training:

* set flag random=1 in your .cfg-file - it will increase precision by training Yolo for different resolutions: [link](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L788)
* increase network resolution in your .cfg-file (height=608, width=608 or any value multiple of 32) - it will increase precision
* check that each object that you want to detect is mandatory labeled in your dataset - no one object in your data set should not be without label. In the most training issues - there are wrong labels in your dataset (got labels by using some conversion script, marked with a third-party tool, ...). Always check your dataset by using: <https://github.com/AlexeyAB/Yolo_mark>
* my Loss is very high and mAP is very low, is training wrong? Run training with -show\_imgs flag at the end of training command, do you see correct bounded boxes of objects (in windows or in files aug\_...jpg)? If no - your training dataset is wrong.
* for each object which you want to detect - there must be at least 1 similar object in the Training dataset with about the same: shape, side of object, relative size, angle of rotation, tilt, illumination. So desirable that your training dataset include images with objects at diffrent: scales, rotations, lightings, from different sides, on different backgrounds - you should preferably have 2000 different images for each class or more, and you should train 2000\*classes iterations or more
* desirable that your training dataset include images with non-labeled objects that you do not want to detect - negative samples without bounded box (empty .txt files) - use as many images of negative samples as there are images with objects
* What is the best way to mark objects: label only the visible part of the object, or label the visible and overlapped part of the object, or label a little more than the entire object (with a little gap)? Mark as you like - how would you like it to be detected.
* for training with a large number of objects in each image, add the parameter max=200 or higher value in the last [yolo]-layer or [region]-layer in your cfg-file (the global maximum number of objects that can be detected by YoloV3 is 0,0615234375\*(width\*height) where are width and height are parameters from [net] section in cfg-file)
* for training for small objects (smaller than 16x16 after the image is resized to 416x416) - set layers = 23 instead of <https://github.com/AlexeyAB/darknet/blob/6f718c257815a984253346bba8fb7aa756c55090/cfg/yolov4.cfg#L895> set stride=4 instead of <https://github.com/AlexeyAB/darknet/blob/6f718c257815a984253346bba8fb7aa756c55090/cfg/yolov4.cfg#L892> and set stride=4 instead of <https://github.com/AlexeyAB/darknet/blob/6f718c257815a984253346bba8fb7aa756c55090/cfg/yolov4.cfg#L989>
* for training for both small and large objects use modified models:
  + Full-model: 5 yolo layers: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3_5l.cfg>
  + Tiny-model: 3 yolo layers: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3-tiny_3l.cfg>
  + YOLOv4: 3 yolo layers: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4-custom.cfg>
* If you train the model to distinguish Left and Right objects as separate classes (left/right hand, left/right-turn on road signs, ...) then for disabling flip data augmentation - add flip=0 here: <https://github.com/AlexeyAB/darknet/blob/3d2d0a7c98dbc8923d9ff705b81ff4f7940ea6ff/cfg/yolov3.cfg#L17>
* General rule - your training dataset should include such a set of relative sizes of objects that you want to detect:
  + train\_network\_width \* train\_obj\_width / train\_image\_width ~= detection\_network\_width \* detection\_obj\_width / detection\_image\_width
  + train\_network\_height \* train\_obj\_height / train\_image\_height ~= detection\_network\_height \* detection\_obj\_height / detection\_image\_height

I.e. for each object from Test dataset there must be at least 1 object in the Training dataset with the same class\_id and about the same relative size:

object width in percent from Training dataset ~= object width in percent from Test dataset

That is, if only objects that occupied 80-90% of the image were present in the training set, then the trained network will not be able to detect objects that occupy 1-10% of the image.

* to speedup training (with decreasing detection accuracy) set param stopbackward=1 for layer-136 in cfg-file
* each: model of object, side, illimination, scale, each 30 grad of the turn and inclination angles - these are *different objects* from an internal perspective of the neural network. So the more *different objects* you want to detect, the more complex network model should be used.
* to make the detected bounded boxes more accurate, you can add 3 parameters ignore\_thresh = .9 iou\_normalizer=0.5 iou\_loss=giou to each [yolo] layer and train, it will increase mAP@0.9, but decrease mAP@0.5.
* Only if you are an **expert** in neural detection networks - recalculate anchors for your dataset for width and height from cfg-file: darknet.exe detector calc\_anchors data/obj.data -num\_of\_clusters 9 -width 416 -height 416 then set the same 9 anchors in each of 3 [yolo]-layers in your cfg-file. But you should change indexes of anchors masks= for each [yolo]-layer, so that 1st-[yolo]-layer has anchors larger than 60x60, 2nd larger than 30x30, 3rd remaining. Also you should change the filters=(classes + 5)\*<number of mask> before each [yolo]-layer. If many of the calculated anchors do not fit under the appropriate layers - then just try using all the default anchors.

1. After training - for detection:

* Increase network-resolution by set in your .cfg-file (height=608 and width=608) or (height=832 and width=832) or (any value multiple of 32) - this increases the precision and makes it possible to detect small objects: [link](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L8-L9)
  + it is not necessary to train the network again, just use .weights-file already trained for 416x416 resolution
  + but to get even greater accuracy you should train with higher resolution 608x608 or 832x832, note: if error Out of memory occurs then in .cfg-file you should increase subdivisions=16, 32 or 64: [link](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L4)

## DLL和SO库封装

* on Linux
  + using build.sh or
  + build darknet using cmake or
  + set LIBSO=1 in the Makefile and do make
* on Windows
  + using build.ps1 or
  + build darknet using cmake or
  + compile build\darknet\yolo\_cpp\_dll.sln solution or build\darknet\yolo\_cpp\_dll\_no\_gpu.sln solution

There are 2 APIs:

* C API: <https://github.com/AlexeyAB/darknet/blob/master/include/darknet.h>
  + Python examples using the C API::
    - <https://github.com/AlexeyAB/darknet/blob/master/darknet.py>
    - <https://github.com/AlexeyAB/darknet/blob/master/darknet_video.py>
* C++ API: <https://github.com/AlexeyAB/darknet/blob/master/include/yolo_v2_class.hpp>
  + C++ example that uses C++ API: <https://github.com/AlexeyAB/darknet/blob/master/src/yolo_console_dll.cpp>

1. To compile Yolo as C++ DLL-file yolo\_cpp\_dll.dll - open the solution build\darknet\yolo\_cpp\_dll.sln, set **x64** and **Release**, and do the: Build -> Build yolo\_cpp\_dll
   * You should have installed **CUDA 10.0**
   * To use cuDNN do: (right click on project) -> properties -> C/C++ -> Preprocessor -> Preprocessor Definitions, and add at the beginning of line: CUDNN;
2. To use Yolo as DLL-file in your C++ console application - open the solution build\darknet\yolo\_console\_dll.sln, set **x64** and **Release**, and do the: Build -> Build yolo\_console\_dll
   * you can run your console application from Windows Explorer build\darknet\x64\yolo\_console\_dll.exe **use this command**: yolo\_console\_dll.exe data/coco.names yolov4.cfg yolov4.weights test.mp4
   * after launching your console application and entering the image file name - you will see info for each object: <obj\_id> <left\_x> <top\_y> <width> <height> <probability>
   * to use simple OpenCV-GUI you should uncomment line //#define OPENCV in yolo\_console\_dll.cpp-file: [link](https://github.com/AlexeyAB/darknet/blob/a6cbaeecde40f91ddc3ea09aa26a03ab5bbf8ba8/src/yolo_console_dll.cpp#L5)
   * you can see source code of simple example for detection on the video file: [link](https://github.com/AlexeyAB/darknet/blob/ab1c5f9e57b4175f29a6ef39e7e68987d3e98704/src/yolo_console_dll.cpp#L75)

yolo\_cpp\_dll.dll-API: [link](https://github.com/AlexeyAB/darknet/blob/master/src/yolo_v2_class.hpp#L42)

struct bbox\_t {

unsigned int x, y, w, h; // (x,y) - top-left corner, (w, h) - width & height of bounded box

float prob; // confidence - probability that the object was found correctly

unsigned int obj\_id; // class of object - from range [0, classes-1]

unsigned int track\_id; // tracking id for video (0 - untracked, 1 - inf - tracked object)

unsigned int frames\_counter;// counter of frames on which the object was detected

};

class Detector {

public:

Detector(std::string cfg\_filename, std::string weight\_filename, int gpu\_id = 0);

~Detector();

std::vector<bbox\_t> detect(std::string image\_filename, float thresh = 0.2, bool use\_mean = false);

std::vector<bbox\_t> detect(image\_t img, float thresh = 0.2, bool use\_mean = false);

static image\_t load\_image(std::string image\_filename);

static void free\_image(image\_t m);

#ifdef OPENCV

std::vector<bbox\_t> detect(cv::Mat mat, float thresh = 0.2, bool use\_mean = false);

std::shared\_ptr<image\_t> mat\_to\_image\_resize(cv::Mat mat) const;

#endif

};

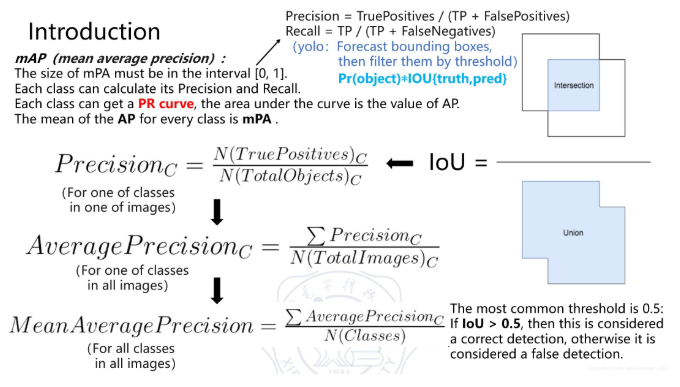
## Command line 使用

对于linux和windows命令行的使用相同：以下以windows为例：

* Yolo v4 COCO - **image**: darknet.exe detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -thresh 0.25
* **Output coordinates** of objects: darknet.exe detector test cfg/coco.data yolov4.cfg yolov4.weights -ext\_output dog.jpg
* Yolo v4 COCO - **video**: darknet.exe detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext\_output test.mp4
* Yolo v4 COCO - **WebCam 0**: darknet.exe detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -c 0
* Yolo v4 COCO for **net-videocam** - Smart WebCam: darknet.exe detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights http://192.168.0.80:8080/video?dummy=param.mjpg
* Yolo v4 - **save result videofile res.avi**: darknet.exe detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights test.mp4 -out\_filename res.avi
* Yolo v3 **Tiny** COCO - video: darknet.exe detector demo cfg/coco.data cfg/yolov3-tiny.cfg yolov3-tiny.weights test.mp4
* **JSON and MJPEG server** that allows multiple connections from your soft or Web-browser ip-address:8070 and 8090: ./darknet detector demo ./cfg/coco.data ./cfg/yolov3.cfg ./yolov3.weights test50.mp4 -json\_port 8070 -mjpeg\_port 8090 -ext\_output
* Yolo v3 Tiny **on GPU #1**: darknet.exe detector demo cfg/coco.data cfg/yolov3-tiny.cfg yolov3-tiny.weights -i 1 test.mp4
* Alternative method Yolo v3 COCO - image: darknet.exe detect cfg/yolov4.cfg yolov4.weights -i 0 -thresh 0.25
* Train on **Amazon EC2**, to see mAP & Loss-chart using URL like: http://ec2-35-160-228-91.us-west-2.compute.amazonaws.com:8090 in the Chrome/Firefox (**Darknet should be compiled with OpenCV**): ./darknet detector train cfg/coco.data yolov4.cfg yolov4.conv.137 -dont\_show -mjpeg\_port 8090 -map
* 186 MB Yolo9000 - image: darknet.exe detector test cfg/combine9k.data cfg/yolo9000.cfg yolo9000.weights
* Remeber to put data/9k.tree and data/coco9k.map under the same folder of your app if you use the cpp api to build an app
* To process a list of images data/train.txt and save results of detection to result.json file use: darknet.exe detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext\_output -dont\_show -out result.json < data/train.txt
* To process a list of images data/train.txt and save results of detection to result.txt use:  
  darknet.exe detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont\_show -ext\_output < data/train.txt > result.txt
* **Pseudo-lableing** - to process a list of images data/new\_train.txt and save results of detection in Yolo training format for each image as label <image\_name>.txt (in this way you can increase the amount of training data) use: darknet.exe detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -thresh 0.25 -dont\_show -save\_labels < data/new\_train.txt
* **To calculate anchors:** darknet.exe detector calc\_anchors data/obj.data -num\_of\_clusters 9 -width 416 -height 416
* To check accuracy mAP@IoU=50: darknet.exe detector map data/obj.data yolo-obj.cfg backup\yolo-obj\_7000.weights
* To check accuracy mAP@IoU=75: darknet.exe detector map data/obj.data yolo-obj.cfg backup\yolo-obj\_7000.weights -iou\_thresh 0.75

## 指标解析

1. MAP指标



1. NMS非极大值抑制



其他

## 其他指令

linux下可利用 ls | sed "s:^:`pwd`/: " >dirpath.txt 得到文件夹下所有文件的全路径。

1、列出当前目录的文件、文件夹完整路径  
   ls -1 |awk '{print i$0}' i=`pwd`'/'

Windows : forfiles /m \*.\* /c “cmd /c echo @path” >a.txt

后台运行方法： nohup darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -map >person.log &

查看日志：cat person.log

实时显示gpu信息：watch -n 0.1 nvidia-smi

Python环境中opencv 读取中文路径存在问题！

# NMS非极大值抑制

