<https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects>

1. [AlexeyAB](https://github.com/AlexeyAB)/[**darknet**](https://github.com/AlexeyAB/darknet)

**How to train (to detect your custom objects):**

(to train **old Yolo v2** yolov2-voc.cfg, yolov2-tiny-voc.cfg, yolo-voc.cfg, yolo-voc.2.0.cfg, ... click by the link)

**Training Yolo v3:**

1. Create file **yolo-obj.cfg** with the same content as in **yolov3.cfg** (or copy yolov3.cfg to yolo-obj.cfg) and:

・change line batch to **batch=64**

・change line subdivisions to **subdivisions=8**

・change line **classes=80** to your number of objects in each of 3 [yolo]-layers:

https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L610

https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L696

https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L783

・change **[filters=255]** to filters=(classes + 5)x3 in the 3 **[convolutional]** before each [yolo] layer

https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L603

https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L689

https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L776

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 **mask** s, 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)**

一般に、フィルタは、クラス、コードおよびマスクの数、すなわちフィルタ=（クラス+コード+1）\* <マスクの数>に依存し、マスクはアンカーのインデックスである。

So for example, for 2 objects, your file **yolo-obj.cfg** should differ from **yolov3.cfg** in such lines in each of 3 [yolo]-layers:

たとえば、2つのオブジェクトの場合、ファイルyolo-obj.cfgは3つの[yolo]レイヤーのそれぞれの行でyolov3.cfgと異なるはずです。

[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

※**labels.txt** に相当

1. 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) 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 for marking bounded boxes of objects and generating annotation files for Yolo v2 & v3: https://github.com/AlexeyAB/**Yolo\_mark**

※**Bounding Box Tool**

It will create **.txt**-file for each .**jpg**-image-file - in the same directory and with the same name, but with **.txt**-extension, and put to file: object number and object coordinates on this image, for each object in new line: **<object-class> <x> <y> <width> <height>**

各.jpg-image-fileの.txt-fileを、同じディレクトリに、同じ名前で作成します。しかし、.txt-extensionを指定して、ファイルに追加します。オブジェクト番号とオブジェクトの座標（このイメージ上の各オブジェクトの改行）：<object-class> <x> <y> <width> <height>

Where:

**<object-class>** - integer number of object from **0** to **(classes-1)**

**<x> <y> <width> <height>** - float values relative to width and height of image, it can be equal from (0.0 to 1.0]

for example: **<x> = <absolute\_x>** / **<image\_width>** or **<height> = <absolute\_height> / <image\_height>**

atention: **<x> <y>** - are center of rectangle (are not top-left corner)

For example for **img1.jpg** you will be created **img1.txt** containing:

1 0.716797 0.395833 0.216406 0.147222

0 0.687109 0.379167 0.255469 0.158333

1 0.420312 0.395833 0.140625 0.166667

1. 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 (154 MB): **https://pjreddie.com/media/files/darknet53.conv.74** and put to the directory **build\darknet\x64**
2. Start training by using the command line: **darknet.exe detector train data/obj.data yolo-obj.cfg darknet53.conv.74**

(file **yolo-obj\_xxx.weights** will be saved to the **build\darknet\x64\backup\** for each 100 iterations) (To disable Loss-Window use **darknet.exe detector train data/obj.data yolo-obj.cfg darknet53.conv.74 -dont\_show,** if you train on computer without monitor like a cloud Amazaon EC2)

1. After training is complete - get result **yolo-obj\_final.weight**s 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 copy **yolo-obj\_2000.weights** from **build\darknet\x64\backup\** to **build\darknet\x64\** and start training using: **darknet.exe detector train data/obj.data yolo-obj.cfg 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.

訓練中に平均値（喪失）フィールドにnanの値が表示されている場合、訓練は間違っていますが、もしnanが他のラインにあれば、訓練は順調に進んでいます。

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

cfgファイルでwidth =またはheight =を変更した場合、新しい幅と高さは32で割り切れる必要があります。

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

訓練の後、次のような検出のためのコマンドを使用する：

**How to train tiny-yolo (to detect your custom objects):**

Do all the same steps as for the full yolo model as described above. With the exception of:

上記のように完全なyoloモデルと同じ手順をすべて実行します。例外：

・Download default weights file for **yolov3-tiny**: **https://pjreddie.com/media/files/yolov3-tiny.weights**

・Get pre-trained weights **yolov3-tiny.conv.15** using command: **darknet.exe partial cfg/yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15**

**実行結果**

[shimatani@bslpc168 ~/darknet]$./darknet partial cfg/yolov3-tiny.cfg bin/yolov3-tiny.weights yolov3-tiny.conv.15 15

layer filters size input output

0 conv 16 3 x 3 / 1 416 x 416 x 3 -> 416 x 416 x 16 0.150 BF

1 max 2 x 2 / 2 416 x 416 x 16 -> 208 x 208 x 16 0.003 BF

2 conv 32 3 x 3 / 1 208 x 208 x 16 -> 208 x 208 x 32 0.399 BF

3 max 2 x 2 / 2 208 x 208 x 32 -> 104 x 104 x 32 0.001 BF

4 conv 64 3 x 3 / 1 104 x 104 x 32 -> 104 x 104 x 64 0.399 BF

5 max 2 x 2 / 2 104 x 104 x 64 -> 52 x 52 x 64 0.001 BF

6 conv 128 3 x 3 / 1 52 x 52 x 64 -> 52 x 52 x 128 0.399 BF

7 max 2 x 2 / 2 52 x 52 x 128 -> 26 x 26 x 128 0.000 BF

8 conv 256 3 x 3 / 1 26 x 26 x 128 -> 26 x 26 x 256 0.399 BF

9 max 2 x 2 / 2 26 x 26 x 256 -> 13 x 13 x 256 0.000 BF

10 conv 512 3 x 3 / 1 13 x 13 x 256 -> 13 x 13 x 512 0.399 BF

11 max 2 x 2 / 1 13 x 13 x 512 -> 13 x 13 x 512 0.000 BF

12 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF

13 conv 256 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 256 0.089 BF

14 conv 512 3 x 3 / 1 13 x 13 x 256 -> 13 x 13 x 512 0.399 BF

15 conv 255 1 x 1 / 1 13 x 13 x 512 -> 13 x 13 x 255 0.044 BF

16 yolo

17 route 13

18 conv 128 1 x 1 / 1 13 x 13 x 256 -> 13 x 13 x 128 0.011 BF

19 upsample 2x 13 x 13 x 128 -> 26 x 26 x 128

20 route 19 8

21 conv 256 3 x 3 / 1 26 x 26 x 384 -> 26 x 26 x 256 1.196 BF

22 conv 255 1 x 1 / 1 26 x 26 x 256 -> 26 x 26 x 255 0.088 BF

23 yolo

Total BFLOPS 5.571

Loading weights from bin/yolov3-tiny.weights...

seen 64

Done!

Saving weights to **yolov3-tiny.conv.15**

[shimatani@bslpc168 ~/darknet]$

・Make your custom model **yolov3-tiny-obj.cfg** based on **cfg/yolov3-tiny\_obj.cfg** instead of **yolov3.cfg**

・Start training: **darknet.exe detector train data/obj.data yolov3-tiny-obj.cfg yolov3-tiny.conv.15**

**実行結果**

[shimatani@bslpc168 ~/darknet]$./darknet detector train cfg/nfpa.data cfg/yolov3-tiny\_obj.cfg yolov3-tiny.conv.15

used slow CUDNN algo without Workspace! Need memory: **4,596,957,184**, available: **1,028,128,768**

CUDNN-slow try to allocate workspace = 2097153 \* sizeof(float), CUDA allocate done!

1309: 0.712015, 0.764630 avg loss, 0.001000 rate, 1.116989 seconds, 83776 images

Loaded: 0.000024 seconds

Region 16 Avg IOU: 0.747966, Class: 0.999759, Obj: 0.380791, No Obj: 0.001220, .5R: 1.000000, .75R: 0.550000, count: 20

Region 23 Avg IOU: 0.644004, Class: 0.999112, Obj: 0.674705, No Obj: 0.000272, .5R: 0.851852, .75R: 0.333333, count: 27

Region 16 Avg IOU: 0.737667, Class: 0.999547, Obj: 0.461520, No Obj: 0.001391, .5R: 1.000000, .75R: 0.520000, count: 25

Region 23 Avg IOU: 0.502956, Class: 0.998191, Obj: 0.588248, No Obj: 0.000161, .5R: 0.533333, .75R: 0.066667, count: 15

1310: 0.764318, 0.764599 avg loss, 0.001000 rate, 1.113713 seconds, 83840 images

Resizing

448 x 448

try to allocate workspace = 308412417 \* sizeof(float), CUDA allocate done!

Loaded: 0.000017 seconds

CUDA Error: out of memory

darknet: ./src/cuda.c:36: check\_error: Assertion `0' failed.

中止 (**コアダンプ**)

**対策**

**Yolov3-tiny\_obj.cfg をチューニング**

batch=64

#subdivisions=2 を

subdivisions=64 に変更してコアダンプ回避（32では回避できず）

学習中の nvidia-smi -l 実行結果

Tue Jun 26 13:16:25 2018

+-----------------------------------------------------------------------------+

| NVIDIA-SMI 390.48 Driver Version: 390.48 |

|-------------------------------+----------------------+----------------------+

| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |

| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |

|===============================+======================+======================|

| 0 GeForce GTX 1080 Off | 00000000:01:00.0 Off | N/A |

| 41% 47C P2 168W / 180W | **7179MiB** / **8119MiB** | 100% Default |

+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+

| Processes: GPU Memory |

| GPU PID Type Process name Usage |

|=============================================================================|

| 0 17172 C ./darknet **7169MiB** |

+-----------------------------------------------------------------------------+

* 残容量＝8119-7179が、availableの値とほぼ一致する

▼2640回目にコアダンプ

Tue Jun 26 13:37:37 2018

+-----------------------------------------------------------------------------+

| NVIDIA-SMI 390.48 Driver Version: 390.48 |

|-------------------------------+----------------------+----------------------+

| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |

| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |

|===============================+======================+======================|

| 0 GeForce GTX 1080 Off | 00000000:01:00.0 Off | N/A |

| 41% 41C P2 39W / 180W | **8045MiB** / **8119MiB** | 0% Default |

+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+

| Processes: GPU Memory |

| GPU PID Type Process name Usage |

|=============================================================================|

| 0 17172 C ./darknet **8035MiB** |

+-----------------------------------------------------------------------------+

**2600**: 0.388150, 0.530272 avg loss, 0.001000 rate, 0.704258 seconds, 166400 images

Saving weights to backup//yolov3-tiny\_obj\_2600.weights

Resizing

288 x 288

try to allocate workspace = 8388609 \* sizeof(float), CUDA allocate done!

Loaded: 0.025257 seconds

Region 16 Avg IOU: 0.790741, Class: 0.998750, Obj: 0.770232, No Obj: 0.005488, .5R: 1.000000, .75R: 0.807692, count: 26

Region 23 Avg IOU: 0.540879, Class: 0.998703, Obj: 0.399075, No Obj: 0.000217, .5R: 0.583333, .75R: 0.083333, count: 12

Region 16 Avg IOU: 0.780422, Class: 0.999775, Obj: 0.921695, No Obj: 0.004973, .5R: 1.000000, .75R: 0.727273, count: 22

Region 23 Avg IOU: 0.434975, Class: 0.999340, Obj: 0.788368, No Obj: 0.000417, .5R: 0.333333, .75R: 0.000000, count: 15

★2600回目までのモデルを使って再学習開始

[shimatani@bslpc168 ~/darknet]$./darknet detector train cfg/nfpa.data cfg/yolov3-tiny\_obj.cfg backup/**yolov3-tiny\_obj\_2600.weights**

▼8390回目でコアダンプ

8389: 0.218512, 0.291891 avg loss, 0.001000 rate, 0.983426 seconds, 536896 images

Loaded: 0.000022 seconds

Region 16 Avg IOU: 0.837159, Class: 0.999924, Obj: 0.725337, No Obj: 0.002841, .5R: 1.000000, .75R: 0.857143, count: 21

Region 23 Avg IOU: 0.720923, Class: 0.999914, Obj: 0.967080, No Obj: 0.000167, .5R: 1.000000, .75R: 0.272727, count: 11

Region 16 Avg IOU: 0.845954, Class: 0.999966, Obj: 0.927195, No Obj: 0.003469, .5R: 1.000000, .75R: 0.939394, count: 33

Region 23 Avg IOU: 0.710273, Class: 0.999931, Obj: 0.977048, No Obj: 0.000147, .5R: 1.000000, .75R: 0.285714, count: 7

8390: 0.169707, 0.279673 avg loss, 0.001000 rate, 0.980634 seconds, 536960 images

Resizing

448 x 448

try to allocate workspace = 308412417 \* sizeof(float), CUDA allocate done!

Loaded: 0.000019 seconds

CUDA Error: out of memory

darknet: ./src/cuda.c:36: check\_error: Assertion `0' failed.

中止 (コアダンプ)

Tue Jun 26 15:39:31 2018

+-----------------------------------------------------------------------------+

| NVIDIA-SMI 390.48 Driver Version: 390.48 |

|-------------------------------+----------------------+----------------------+

| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |

| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |

|===============================+======================+======================|

| 0 GeForce GTX 1080 Off | 00000000:01:00.0 Off | N/A |

| 40% 40C P2 38W / 180W | 8051MiB / 8119MiB | 0% Default |

+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+

| Processes: GPU Memory |

| GPU PID Type Process name Usage |

|=============================================================================|

| 0 6409 C ./darknet 8041MiB |

+-----------------------------------------------------------------------------+

Tue Jun 26 15:39:36 2018

+-----------------------------------------------------------------------------+

| NVIDIA-SMI 390.48 Driver Version: 390.48 |

|-------------------------------+----------------------+----------------------+

| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |

| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |

|===============================+======================+======================|

| 0 GeForce GTX 1080 Off | 00000000:01:00.0 Off | N/A |

| 40% 40C P0 49W / 180W | 0MiB / 8119MiB | 0% Default |

+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+

| Processes: GPU Memory |

| GPU PID Type Process name Usage |

|=============================================================================|

| No running processes found |

+-----------------------------------------------------------------------------+

---

★8300回目までのモデルを使って再学習

./darknet detector train cfg/nfpa.data cfg/yolov3-tiny\_obj.cfg backup/**yolov3-tiny\_obj\_8300.weights**

▼9850回目でコアダンプ

★9800回目までのモデルを使って再学習

▼10820回目でコアダンプ

10820: 0.154024, 0.243790 avg loss, 0.001000 rate, 0.870723 seconds, 692480 images

Resizing

448 x 448

try to allocate workspace = 308412417 \* sizeof(float), CUDA allocate done!

Loaded: 0.000020 seconds

CUDA Error: out of memory

darknet: ./src/cuda.c:36: check\_error: Assertion `0' failed.

中止 (コアダンプ)

★10100回までのモデルを使ってモデル変換

(tensorflow) [shimatani@bslpc168 ~/darkflow]$./flow --model cfg/yolov3-tiny\_obj.cfg --load bin/yolov3-tiny\_obj\_10100.weights --savepb

▼エラー

/home/shimatani/darkflow/darkflow/dark/darknet.py:54: UserWarning: ./cfg/yolov3-tiny\_obj\_10100.cfg not found, use cfg/yolov3-tiny\_obj.cfg instead

cfg\_path, FLAGS.model))

Parsing cfg/yolov3-tiny\_obj.cfg

Layer [yolo] not implemented

(tensorflow) [shimatani@bslpc168 ~/darkflow]$

言われる通り yolov3-tiny\_obj\_10100.cfg を yolov3-tiny\_obj.cfg にリネームして再実行

(tensorflow) [shimatani@bslpc168 ~/darkflow]$./flow --model cfg/yolov3-tiny\_obj.cfg --load bin/yolov

3-tiny\_obj.weights --savepb

▼エラー

Parsing ./cfg/yolov3-tiny\_obj.cfg

Layer [yolo] not implemented

★github　pjreddie/darknet　で質問

**How to train tiny YOLO? #517**

<https://github.com/pjreddie/darknet/issues/517>

[](https://github.com/keides2)

フォームの始まり

フォームの終わり

1. [**keides2**](https://github.com/keides2)**commented 11 hours ago**

|  |
| --- |
| Hi [**@AlexeyAB**](https://github.com/AlexeyAB) , I trained with yolov3-tiny\_obj.cfg and yolov3-tiny.weights with the following command. $ ./darknet partial cfg / yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15 $ ./darknet detector train data / obj.data yolov3-tiny\_obj.cfg yolov3-tiny.conv.15  I want to convert the generated .weights file to pb file using darkflow's following command and run it on android mobile phone, but I got an error. (I will add the details of the error tomorrow) $ ./flow --module cfg/yolov3-tiny\_obj.cfg --load bin/yolov3-tiny\_obj-10000.weights --savepb  Can darkflow convert yolov3-tiny\_obj\_10000.weights to pb file?  If possible, can I use this pb file with android-yolo-v2-master?  Addition: (tensorflow) [shimatani@bslpc168 ~/darkflow]$./flow --model cfg/yolov3-tiny\_obj.cfg --load bin/yolov 3-tiny\_obj\_10000.weights --savepb  /home/shimatani/darkflow/darkflow/dark/darknet.py:54: UserWarning: ./cfg/yolov3-tiny\_obj\_10000.cfg not found, use cfg/yolov3-tiny\_obj.cfg instead cfg\_path, FLAGS.model)) Parsing cfg/yolov3-tiny\_obj.cfg Layer [yolo] not implemented  Correction: If possible, can I use this pb file with android-yolo-v2 of szaza? |

フォームの始まり

フォームの終わり

[](https://github.com/AlexeyAB)

1. [**AlexeyAB**](https://github.com/AlexeyAB)**commented**[**an hour ago**](https://github.com/pjreddie/darknet/issues/517#issuecomment-400502840)

|  |
| --- |
| [**@keides2**](https://github.com/keides2)  As I see <https://github.com/szaza/android-yolo-v2> and <https://github.com/thtrieu/darkflow> don't support Yolo v3. It supports only Yolo v2. |

フォームの始まり

+1 1

フォームの終わり

[](https://github.com/keides2)

フォームの始まり

フォームの終わり

1. [**keides2**](https://github.com/keides2)**commented**[**an hour ago**](https://github.com/pjreddie/darknet/issues/517#issuecomment-400503538)

|  |
| --- |
| I got it. Thank you so much, [**@AlexeyAB**](https://github.com/AlexeyAB) . |

For training Yolo based on other models (DenseNet201-Yolo or ResNet50-Yolo), you can download and get pre-trained weights as showed in this file: [**https://github.com/AlexeyAB/darknet/blob/master/build/darknet/x64/partial.cmd**](https://github.com/AlexeyAB/darknet/blob/master/build/darknet/x64/partial.cmd)

If you made you custom model that isn't based on other models, then you can train it without pre-trained weights, then will be used random initial weights.

他のモデル（DenseNet201-YoloまたはResNet50-Yolo）に基づいてYoloをトレーニングする場合、このファイルに示されているように、事前にトレーニングされたウェイトをダウンロードして入手できます：

他のモデルに基づいていないカスタムモデルを作成した場合は、事前にトレーニングされたウェイトを使用せずにトレーニングしてから、初期ウェイトをランダムに使用します。

1. **When should I stop training:**

Usually sufficient 2000 iterations for each class(object). But for a more precise definition when you should stop training, use the following manual:

通常は各クラス（オブジェクト）に対して2000回の反復が必要です。しかし、トレーニングをやめるべきときのより正確な定義については、以下のマニュアルを使用してください：

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

トレーニング中は、様々なエラーのインジケータが表示されますが、これ以上減少しない場合は停止する必要があります。0.XXXXXXX平均：

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, **0.060730 avg**, 0.001000 rate, 3.868000 seconds, 576128 images Loaded: 0.000000 seconds

* **9002** - iteration number (number of batch)
* **0.060730 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.

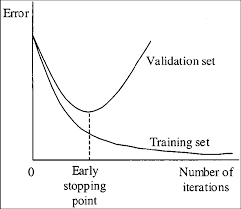
多くの反復で平均損失0.xxxxxx avgが減少することがなくなり次第、トレーニングを中止する必要があります。

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:

訓練が止まったら、darknet \ build \ darknet \ x64 \ backupの最後のウェイトファイルをとり、それらの中からベストを選ぶべきです：

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**:

たとえば、9000回の反復後にトレーニングを中止しましたが、最良の結果が以前の重み（7000,8000,9000）のいずれかを与えることができます。オーバーフィットのために起こる可能性があります。オーバーフィット - トレーニングデータセットから画像上のオブジェクトを検出できますが、他の画像上のオブジェクトは検出できません。あなたは早期停止点からweightsを得るべきです：

[](https://camo.githubusercontent.com/51af5be5cfa94b6d741c90d10a163b168bf9170e/68747470733a2f2f6873746f2e6f72672f66696c65732f3564632f3761652f3766612f35646337616537666164396434653365623361343834633538626663316666352e706e67)

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):

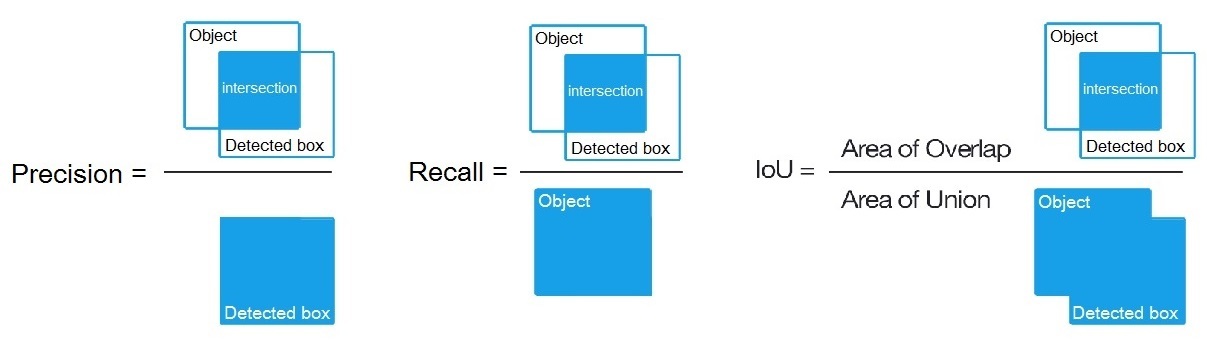
Choose weights-file **with the highest IoU** (intersect of union) and mAP (mean average precision)

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

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

* **IoU** (intersect of union) - average instersect of 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**.

[](https://camo.githubusercontent.com/ffd00e8c7f54d4710edea3bb47e201c8bedab074/68747470733a2f2f6873746f2e6f72672f66696c65732f6361382f3836362f6437362f63613838363664373666623834303232383934306462663434326137663036612e6a7067)

1. **How to calculate mAP on PascalVOC 2007:**
2. To calculate mAP (mean average precision) on PascalVOC-2007-test:

* Download PascalVOC dataset, install Python 3.x and get file 2007\_test.txt as described here: <https://github.com/AlexeyAB/darknet#how-to-train-pascal-voc-data>
* Then download file <https://raw.githubusercontent.com/AlexeyAB/darknet/master/scripts/voc_label_difficult.py> to the dir build\darknet\x64\data\ then run voc\_label\_difficult.py to get the file difficult\_2007\_test.txt
* Remove symbol # from this line to un-comment it: <https://github.com/AlexeyAB/darknet/blob/master/build/darknet/x64/data/voc.data#L4>
* Then there are 2 ways to get mAP:
  1. Using Darknet + Python: run the file build/darknet/x64/calc\_mAP\_voc\_py.cmd - you will get mAP for yolo-voc.cfgmodel, mAP = 75.9%
  2. Using this fork of Darknet: run the file build/darknet/x64/calc\_mAP.cmd - you will get mAP for yolo-voc.cfg model, mAP = 75.8%

(The article specifies the value of mAP = 76.8% for YOLOv2 416×416, page-4 table-3: <https://arxiv.org/pdf/1612.08242v1.pdf>. We get values lower - perhaps due to the fact that the model was trained on a slightly different source code than the code on which the detection is was done)

* if you want to get mAP for tiny-yolo-voc.cfg model, then un-comment line for tiny-yolo-voc.cfg and comment line for yolo-voc.cfg in the .cmd-file
* if you have Python 2.x instead of Python 3.x, and if you use Darknet+Python-way to get mAP, then in your cmd-file use reval\_voc.py and voc\_eval.py instead of reval\_voc\_py3.py and voc\_eval\_py3.py from this directory: <https://github.com/AlexeyAB/darknet/tree/master/scripts>

1. **Custom object detection:**

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

| **[Yolo_v2_training](https://camo.githubusercontent.com/a40a72a25b947d7ba07a2f8b35ae9fb09aa500be/68747470733a2f2f6873746f2e6f72672f66696c65732f6431322f3165372f3531352f64313231653735313566366134656236393439313366313064653566326236312e6a7067)** | **[Yolo_v2_training](https://camo.githubusercontent.com/ebd8a1b7078ebac176cd45fdcee578c0d2a607ae/68747470733a2f2f6873746f2e6f72672f66696c65732f3732372f6337652f3565392f37323763376535653939626634643461613334303237626236613565346261622e6a7067)** |
| --- | --- |

1. **How to improve object detection:**
2. 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
* 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
* check that each object are 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>
* 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 images for each class 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
* for training with a large number of objects in each image, add the parameter max=200 or higher value in the last layer [region] in your cfg-file
* for training for small objects - set layers = -1, 11 instead of <https://github.com/AlexeyAB/darknet/blob/6390a5a2ab61a0bdf6f1a9a6b4a739c16b36e0d7/cfg/yolov3.cfg#L720> and set stride=4 instead of <https://github.com/AlexeyAB/darknet/blob/6390a5a2ab61a0bdf6f1a9a6b4a739c16b36e0d7/cfg/yolov3.cfg#L717>
* 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
* to speedup training (with decreasing detection accuracy) do Fine-Tuning instead of Transfer-Learning, set param stopbackward=1 here: <https://github.com/AlexeyAB/darknet/blob/6d44529cf93211c319813c90e0c1adb34426abe5/cfg/yolov3.cfg#L548>

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)
  + you do not need to train the network again, just use .weights-file already trained for 416x416 resolution
  + 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)

1. **How to mark bounded boxes of objects and create annotation files:**

Here you can find repository with GUI-software for marking bounded boxes of objects and generating annotation files for Yolo v2 & v3: <https://github.com/AlexeyAB/Yolo_mark>

With example of: train.txt, obj.names, obj.data, yolo-obj.cfg, air1-6.txt, bird1-4.txt for 2 classes of objects (air, bird) and train\_obj.cmd with example how to train this image-set with Yolo v2 & v3

1. **Using Yolo9000**

Simultaneous detection and classification of 9000 objects:

* yolo9000.weights - (186 MB Yolo9000 Model) requires 4 GB GPU-RAM: <http://pjreddie.com/media/files/yolo9000.weights>
* yolo9000.cfg - cfg-file of the Yolo9000, also there are paths to the 9k.tree and coco9k.map<https://github.com/AlexeyAB/darknet/blob/617cf313ccb1fe005db3f7d88dec04a04bd97cc2/cfg/yolo9000.cfg#L217-L218>
  + 9k.tree - **WordTree** of 9418 categories - <label> <parent\_it>, if parent\_id == -1 then this label hasn't parent: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/build/darknet/x64/data/9k.tree>
  + coco9k.map - map 80 categories from MSCOCO to WordTree 9k.tree: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/build/darknet/x64/data/coco9k.map>
* combine9k.data - data file, there are paths to: 9k.labels, 9k.names, inet9k.map, (change path to your combine9k.train.list): <https://raw.githubusercontent.com/AlexeyAB/darknet/master/build/darknet/x64/data/combine9k.data>
  + 9k.labels - 9418 labels of objects: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/build/darknet/x64/data/9k.labels>
  + 9k.names - 9418 names of objects: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/build/darknet/x64/data/9k.names>
  + inet9k.map - map 200 categories from ImageNet to WordTree 9k.tree: <https://raw.githubusercontent.com/AlexeyAB/darknet/master/build/darknet/x64/data/inet9k.map>

1. **How to use Yolo as DLL**
2. To compile Yolo as C++ DLL-file yolo\_cpp\_dll.dll - open in MSVS2015 file build\darknet\yolo\_cpp\_dll.sln, set **x64**and **Release**, and do the: Build -> Build yolo\_cpp\_dll
   * You should have installed **CUDA 9.1**
   * To use cuDNN do: (right click on project) -> properties -> C/C++ -> Preprocessor -> Preprocessor Definitions, and add at the beginning of line: CUDNN;
3. To use Yolo as DLL-file in your C++ console application - open in MSVS2015 file 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
   * or you can run from MSVS2015 (before this - you should copy 2 files yolo-voc.cfg and yolo-voc.weights to the directory build\darknet\ )
   * 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)

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);

#endif

};