Multi Class Object Detection - Yolov3

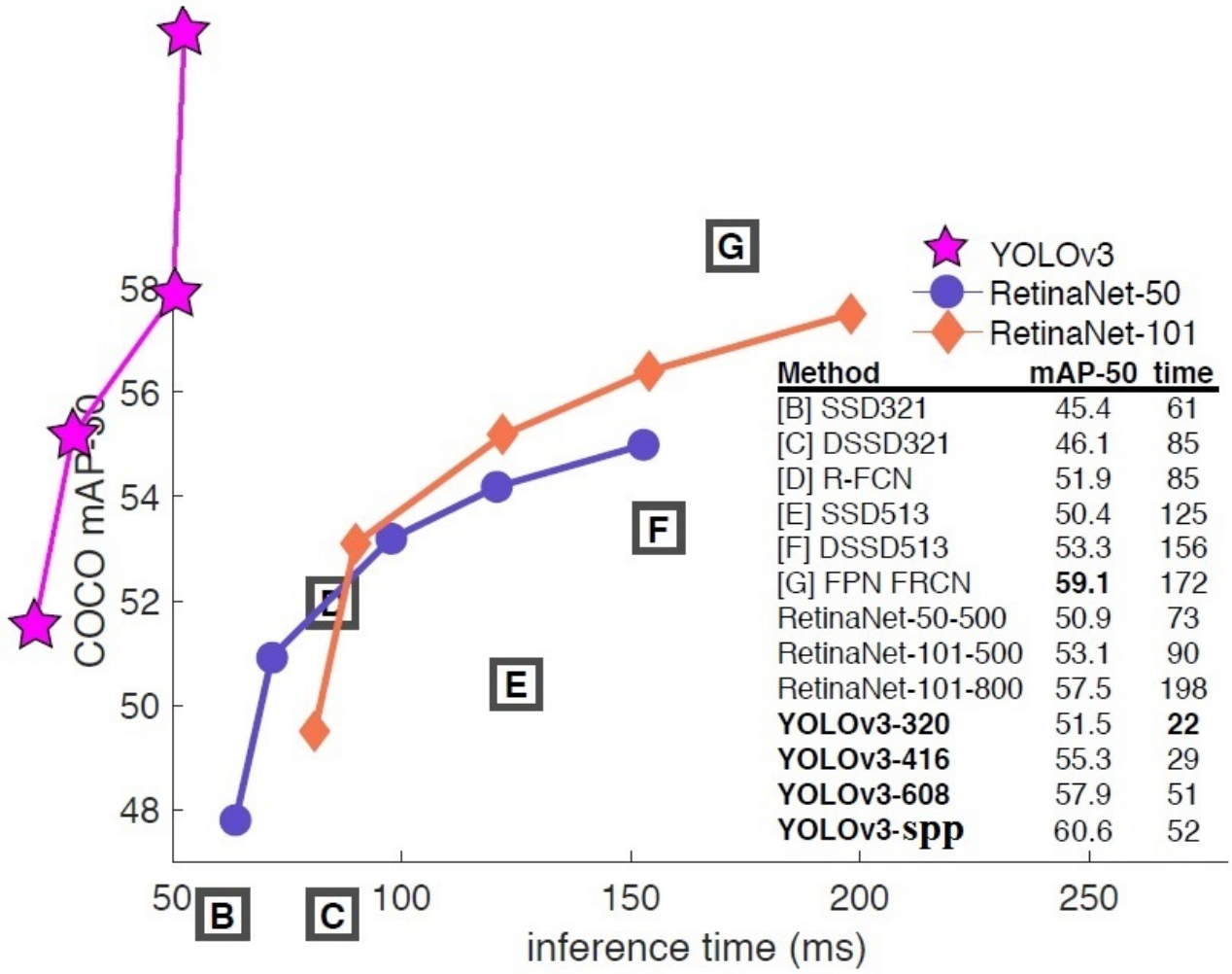
Developer - Kaustav Vats

Contents

Model Info

- Darknet-Yolov3 https://github.com/AlexeyAB/darknet
- Git clone this repo. Yolov3 is much faster than other models like FRCNN, SSD MobileNet V1, V2. |

|



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51
YOLOv3-spp	60.6	52

Figure 3. Again adapted from the [7], this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy → [14].

mAP@0.5 (AP50) https://pjreddie.com/media/files/papers/YOLOv3.pdf | |---|---|

Yolov3 is the state of the art model right now for object detection. There's a tradeoff between `Accuracy` & `time`. We read many blogs regarding which model performs best in both. Yolov3 was most suitable model for our application.

VM Instance Details (GCP Setup)

1. Machine type: `n1-standard-4 (4 vCPUs, 15 GB memory)`
2. GPU: 1 x `NVIDIA Tesla V100`
3. Custom metadeta: `install-nvidia-driver`:`True` (Key, value)
4. OS: `Ubuntu 16.04LTS`, `Boot size - 100 GB`,

Note:- Use template mentioned in this link.
[https://console.cloud.google.com/compute/instanceTemplates/details/deep-learning-template?project=multiclassobjectdetection]

Requirements

- **CMake >= 3.8** for modern CUDA support: https://cmake.org/download/
- **CUDA 10.0**: https://developer.nvidia.com/cuda-toolkit-archive (on Linux do Post-installation Actions)
- **OpenCV >= 2.4**: use your preferred package manager (brew, apt), build from source using vcpkg or download from OpenCV official site (on Windows set system variable `OpenCV_DIR` = `C:\opencv\build`

- where are the `include` and `x64` folders image)
- **cuDNN >= 7.0 for CUDA 10.0** https://developer.nvidia.com/rdp/cudnn-archive (on **Linux** copy `cudnn.h`,`libcudnn.so`... as desribed here https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#installlinux-tar , on **Windows** copy `cudnn.h`,`cudnn64_7.dll`, `cudnn64_7.lib` as desribed here https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#installwindows)
- **GPU with CC >= 3.0**: https://en.wikipedia.org/wiki/CUDA#GPUs_supported
- on Linux **GCC or Clang**, on Windows **MSVC 2015/2017/2019** https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community

Note:- If you are facing any issue regarding installation of NVIDIA GPU Drivers and Tool Kit, Refer to this link.

Dataset details

Classes - [Face, Vehicle(Car, Bicycle, Truck, Airplane ...)]
To create your own custom dataset with above mentioned classes, you first need to collect dataset for these classes and then convert these dataset annotations into Yolov3 format.
`<object-class> <x_center> <y_center> <width> <height>`

Where:

- `<object-class>` - integer object number from `0` to `(classes-1)`
- `<x_center> <y_center> <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_center> <y_center>` - 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. **Face Dataset** - For face dataset I used WIDER FACE Dataset, this dataset contains enough face images for training. It contains images of various sizes of face and varying number of faces in each image. Given annotains format for WIDER FACE is

```
0--Parade/0_Parade_marchingband_1_465.jpg  // Image Name
1                                          // No of face present in an image.
345 211 4 4 2 0 0 0 2 0                    // Face 1
0--Parade/0_Parade_Parade_0_913.jpg
2
238 146 212 246 0 0 1 0 0 0
612 192 206 234 0 0 1 0 0 0
```

Each text file contains 1 row per detected bounding box, in the format `[left, top, width,height ...]`.
I wrote a script to convert this representation for yolov3.\

- Download WIDER FACE Dataset and extract this dataset into this path `WiderFace-Dataset/`.
- run this command while you are present int `WiderFace-Dataset/` directory, `python3 convert.py`.
- This will create labels of wider face dataset and output them in `face-vehicle/` directory.
- Copy and paste all images from `WiderFace-Dataset/Wider_train` and `WiderFace-Dataset/Wider_val` to `face-vehicle/`.
- It will also create a list of train images and test images in `WiderFace-Dataset/` with names `train.txt` and `test.txt`.

2. **Vehicle Dataset** - For vehicle dataset I used COCO Dataset, I extracted 6 class Images for creating our own custom dataset which contains objects of classes [Bicycle, Car, Motorcycle, Airplane, Bus, Truck] where as Vehicle is super class of these classes. To understand annotation of COCO dataset refer to this link.
   I extracted 6 classes from coco dataset and created a new dataset into yolov3 format.

- Create `Train` and `Test` folder inside `CocoVehiclePrep/TxTcoco/`.
- run `CocoLabels.py` in `CocoVehiclePrep/` directory.
- This will create Labels in `TxTcoco/Train` and `TxTcoco/Test`.
- It will also create a list of train images and test images in `CocoVehiclePrep/` with names `train.txt` and `test.txt`.

Extra- If you are not using any existing data and want to create your own custom data. Use LabelImg tool to label data for Yolo and Pascal VOC format.

Pre-trained models

1. Download pre-trained weights for the convolutional layers (154 MB): http://pjreddie.com/media/files/darknet53.conv.74 and put to the directory `build\darknet\x64`

Note- To retrain model on similar class objects. Use weight file present in `/backup/yolov3-face-vehicle_last.weights`.

How to compile on Linux

1. First you need to compile darknet on your system. Follow the steps mentioned in https://github.com/AlexeyAB/darknet#how-to-compile-on-linux
2. To compile darknet on your system, you should enable these options.

- `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

All the above options are neccessary to use GPU and also to visualize results while training. I didn't compiled it with OpenCV and faced lot of issues regarding training and also wasn't able to fully observer the proceedure.

How to train (to detect your custom object)

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 max_batches to (`classes*2000`), f.e. `max_batches=6000` if you train for 3 classes
- change line steps to 80% and 90% of max_batches, f.e. `steps=4800,5400`
- 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`)

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:

```
[convolutional]
filters=21

[region]
classes=2
```

2. Create file `obj.names` in the directory `build\darknet\x64\data\`, with objects names - each in new line

3. 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/
```

4. Put image-files (.jpg) of your objects in the directory `build\darknet\x64\data\obj\`

5. 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

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_center> <y_center> <width> <height>`

Where:

- `<object-class>` - integer object number from `0` to `(classes-1)`
- `<x_center> <y_center> <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_center> <y_center>` - 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
```

6. 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
```

7. 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`

8. Start training by using the command line: `darknet.exe detector train data/obj.data yolo-obj.cfg darknet53.conv.74`

   To train on Linux use command: `./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74` (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 darknet53.conv.74 -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 darknet53.conv.74 -dont_show -mjpeg_port 8090 -map` then open URL `http://ip-address:8090` in Chrome/Firefox browser)

8.1. 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 darknet53.conv.74 -map`

9. 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

How to re-train on new data

1. Add new data as mentioned in dataset section.
2. Update train and test list with new images.
3. Update cfg with instructions mentioned in training section. (update number of classes, filters, steps).
4. while training use weight present in `backup/` directory.

How to improve object detection

1. Before training:

- set flag `random=1` in your `.cfg`-file - it will increase precision by training Yolo for different resolutions: link

- 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

- 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

- 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.

2. 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

    - 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

How to test the model

On Linux use `./darknet` instead of `darknet.exe`, like this:`./darknet detector test ./cfg/coco.data ./cfg/yolov3.cfg ./yolov3.weights`

On Linux find executable file `./darknet` in the root directory, while on Windows find it in the directory `\build\darknet\x64`

- Yolo v3 COCO - **image**: `darknet.exe detector test cfg/coco.data cfg/yolov3.cfg yolov3.weights -thresh 0.25`
- **Output coordinates** of objects: `darknet.exe detector test cfg/coco.data yolov3.cfg yolov3.weights -ext_output dog.jpg`
- Yolo v3 COCO - **video**: `darknet.exe detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -ext_output test.mp4`
- Yolo v3 COCO - **WebCam 0**: `darknet.exe detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -c 0`
- Yolo v3 COCO for **net-videocam** - Smart WebCam: `darknet.exe detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights http://192.168.0.80:8080/video?dummy=param.mjpg`
- Yolo v3 - **save result videofile res.avi**: `darknet.exe detector demo cfg/coco.data cfg/yolov3.cfg yolov3.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/yolov3.cfg yolov3.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 yolov3.cfg darknet53.conv.74 -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/yolov3.cfg yolov3.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/yolov3.cfg yolov3.weights -dont_show -ext_output < data/train.txt > result.txt`
- Pseudo-labeling - 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/yolov3.cfg yolov3.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`

How to extract inference model

1. All models are stored in backup folders, so copy the latest model, or your final model.

Improvements

1. Compile darknet with OpenCV for better visualization and testing of the model. This is also required for better testing of the model.
2. Improve script of extracting coco dataset, modify script such that it only extracts images of object present in it. It shouldn't contain any other object that was present in original coco dataset.

For more Information refer AlexAB/Darknet/README.md

How I run darknet code on GCP

~/multi-obj-det/darknet/darknet detector train data/face-vehicle.data cfg/yolov3-face-vehicle.cfg backup/yolov3-face-vehicle_last.weights