如何在无人机上部署YOLOv4物体检测器

* 代码编译
    * [准备工作（如何安装依赖项）](#准备工作)
    * [在Linux上如何编译](#Linux上编译)
    * [常见编译问题](#常见编译问题)
* 运行代码
    * [预训练模型](#预训练模型)
    * [运行指令介绍](#运行指令介绍)
* 如何训练
    * [如何构建自己的训练数据](#如何构建自己的训练数据)
    * [开始训练（训练相关指令）](#开始训练YOLO)
    * [训练YOLOv3-Tiny](#训练YOLOv3-Tiny)
    * [多GPU训练](#多GPU训练)
    * [训练常见程序问题](#训练常见程序问题)
    * [何时应该停止训练](#何时应该停止训练)
    * [如何提升检测效果](#如何提升检测效果)
* 如何将训练好的模型部署到无人机上
    * [TX2上的准备工作](#TX2上的准备工作)
    * [部署Darknet-ROS](#部署Darknet-ROS)

准备工作

* 推荐使用**Ubuntu 18.04**
* **CMake >= 3.8**: https://cmake.org/download/
* **CUDA >= 10.0**: https://developer.nvidia.com/cuda-toolkit-archive
* **OpenCV >= 2.4**: https://opencv.org/releases.html
* **cuDNN >= 7.0 for CUDA >= 10.0** https://developer.nvidia.com/rdp/cudnn-archive
* **GPU with CC >= 3.0**: https://en.wikipedia.org/wiki/CUDA#GPUs_supported
* **GCC**

Linux上编译

下载YOLOv4源码，推荐使用**Ubuntu 18.04**：

```
sudo apt-get install -y git
git clone https://github.com/AlexeyAB/darknet.git
```

> 配置`Makefile`文件中的参数，然后运行`make -j8`进行编译，具体参数解释如下：

* `GPU=1` 使用CUDA和GPU（CUDA默认路径为`/usr/local/cuda`）
* `CUDNN=1`使用cuDNN v5-v7加速网络（cuDNN默认路径`/usr/local/cudnn`）
* `CUDNN_HALF=1` 使用Tensor Cores（可用GPU为Titan V / Tesla V100 / DGX-2或者更新的）检测速度3x，训练速度2x
* `OPENCV=1` 使用OpenCV 4.x/3.x/2.4.x，运行检测视频和摄像机
* `DEBUG=1` 编译调试版本
* `OPENMP=1` 使用OpenMP利用多CPU加速
* `LIBSO=1` 编译`darknet.so`
  * 使用`uselib`来运行YOLO，输入指令如下：
`LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH ./uselib test.mp4`
  * 在自己的代码中嵌入YOLO，请参考例程：
https://github.com/AlexeyAB/darknet/blob/master/src/yolo_console_dll.cpp

* `ZED_CAMERA=1` 增加ZED-3D相机的支持（需要先安装好ZED SDK）
  * 运行`LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH ./uselib data/coco.names cfg/yolov4.cfg yolov4.weights zed_camera`

常见编译问题

> /bin/sh: 1: nvcc: not found

首先确保CUDA正确安装，并且在路径`/usr/local/cuda`下，然后输入如下指令：

```
echo "PATH=/usr/local/cuda/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
```

> include/darknet.h:46:10: fatal error: cudnn.h: No such file or directory

首先下载`cuDNN`，https://developer.nvidia.com/rdp/cudnn-archive，需要根据自己的CUDA版本选择，然后解压，输入指令：

```
sudo cp -r cudnn-10.1-linux-x64-v7.6.5.32/cuda /usr/local/cudnn
```

预训练模型

所有模型都是在MS-COCO数据集上训练，模型包括两个文件（`cfg`和`weights`）

R表示在RTX 2070设备上的FPS，**V**表示在Tesla V100设备上的FPS

> **百度网盘**打包下载，链接：https://pan.baidu.com/s/1QQPB27n18XeRDnhHA2Gxuw，提取码：uill

* [yolov4.cfg](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4.cfg) - 245 MB: [yolov4.weights](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights)
    * `width=608 height=608`: **65.7 AP@0.5 | 43.5 AP@0.5:0.95 - 34(R) FPS / 62(V) FPS** - 128.5 BFlops
    * `width=512 height=512`: **64.9 AP@0.5 | 43.0 AP@0.5:0.95 - 45(R) FPS / 83(V) FPS** - 91.1 BFlops
    * `width=416 height=416`: **62.8 AP@0.5 | 41.2 AP@0.5:0.95 - 55(R) FPS / 96(V) FPS** - 60.1 BFlops
    * `width=320 height=320`: **60.0 AP@0.5 | 38.0 AP@0.5:0.95 - 63(R) FPS / 123(V) FPS** - 35.5 BFlops

* [yolov3-tiny-prn.cfg](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3-tiny-prn.cfg) - 18.8 MB: [yolov3-tiny-prn.weights](https://drive.google.com/file/d/18yYZWyKbo4XSDVyztmsEcF9B_6bxrhUY/view?usp=sharing)
  * `width=416 height=416`: **33.1 AP@0.5 - 370(R) FPS** - 3.5 BFlops

* [enet-coco.cfg (EfficientNetB0-Yolov3)](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/enet-coco.cfg) - 18.3 MB: [enetb0-coco_final.weights](https://drive.google.com/file/d/1FlHeQjWEQVJt0ay1PVsiuuMzmtNyv36m/view)
  * `width=416 height=416`: **45.5 AP@0.5 - 55(R) FPS** - 3.7 BFlops

* [csresnext50-panet-spp-original-optimal.cfg](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/csresnext50-panet-spp-original-optimal.cfg) - 217 MB: [csresnext50-panet-spp-original-optimal_final.weights](https://drive.google.com/open?id=1_NnfVgj0EDtb_WLNoXV8Mo7WKgwdYZCc)
  * `width=608 height=608`: **65.4 AP@0.5 | 43.2 AP@0.5:0.95 - 32(R) FPS** - 100.5 BFlops

```
*    [yolov3-spp.cfg](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3-spp.cfg)  -  240  MB:  [yolov3-spp.weights](https://pjreddie.com/media/files/yolov3-spp.weights)
  * `width=608 height=608`: **60.6 AP@0.5 - 38(R) FPS** - 141.5 BFlops

*        [yolov3.cfg](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3.cfg)    -    236    MB:    [yolov3.weights](https://pjreddie.com/media/files/yolov3.weights)
  * `width=416 height=416`: **55.3 AP@0.5 - 66(R) FPS** - 65.9 BFlops

*   [yolov3-tiny.cfg](https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3-tiny.cfg) - 33.7 MB: [yolov3-tiny.weights](https://pjreddie.com/media/files/yolov3-tiny.weights)
  * `width=416 height=416`: **33.1 AP@0.5 - 345(R) FPS** - 5.6 BFlops
```

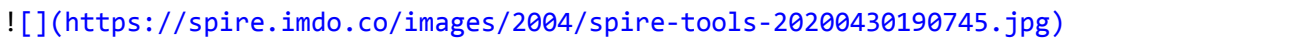
可以在如下路径找到所有的cfg文件：`darknet/cfg/`

运行指令介绍

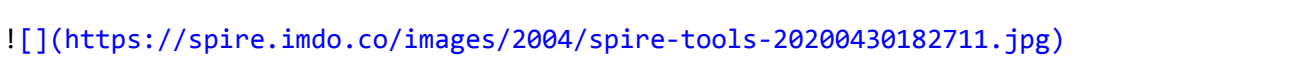
需要将训练好的`weights`文件放到`darknet`根目录下，运行如下指令：

* 检测单张图像
```
./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -thresh 0.25
```
* 检测给定路径的单张图像（参数最后的路径需要写待检测图像的路径）
```
./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext_output /home/jario/Pictures/h1.jpg
```
* 检测给定路径的单个视频
```
./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext_output test.mp4
```
* 检测给定路径的单个视频，并将检测结果保存为视频
```
./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights test.mp4 -out_filename res.avi
```
* 利用摄像机实时检测（YOLOv4）
```
./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -c 0
```
* 利用摄像机实时检测（YOLOv3-Tiny）
```
./darknet detector demo cfg/coco.data cfg/yolov3-tiny.cfg yolov3-tiny.weights -c 0
```
* 在GPU1上检测给定路径的单个视频
```
./darknet detector demo cfg/coco.data cfg/yolov3-tiny.cfg yolov3-tiny.weights -i 1 test.mp4
```
* 检测列表`data/train.txt`中图像，并将结果保存在`result.json`
```
./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext_output -dont_show -out result.json < data/train.txt
```

```
* 检测列表`data/train.txt`中图像，并将结果保存在`result.txt`
```
./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show -ext_output < data/train.txt > result.txt
```

### 如何构建自己的训练数据

下载数据集标注工具，下载地址：(https://pan.baidu.com/s/1EE52cDStjIxsRgM_a9pWQQ) (password: 4b2q) 或者 [**Spire Web**](http://121.36.68.10/tools/ImageLabelTools-4.1.7.zip).

数据集管理软件github地址：https://github.com/jario-jin/spire-image-manager

#### 打开标注软件 SpireImageTools_x.x.x.exe

首先点击Tools->Setting...，填写一个 save path (所有的标注文件都会存储在这个文件夹中)

![](https://spire.imdo.co/images/2004/spire-tools-save-path.jpg)

如果采集的数据集是视频 (**如果采集的是图像，则调过这一步骤**)，点击 Input->Video， 选择要标注的视频。

![](https://spire.imdo.co/images/2004/spire-tools-open-video.jpg)

然后，点击Tools->Video to Image

![](https://spire.imdo.co/images/2004/spire-tools-image2video.jpg)

点击OK 后，等待完成，结果会存储在

![](https://spire.imdo.co/images/2004/spire-tools-20200430182601.jpg)

#### 打开需要标注的图像

Input->Image Dir， 找到需要标注的图像所在文件夹 Ctrl+A，全选，打开

![](https://spire.imdo.co/images/2004/spire-tools-20200430182620.jpg)

点击，Tools->Annotate Image->Instance Label，开始标注图像

![](https://spire.imdo.co/images/2004/spire-tools-start-label.jpg)

在 label 中填写待标注目标名称，然后将对话框拖到一边
在主窗口中开始标注，鼠标滚轮放大缩小图像，按住左键移动可视图像区域不断点击左键将目标框包围，使用 Yolo 训练时，点击 2 个点即可

![](https://spire.imdo.co/images/2004/spire-tools-double-click.jpg)

标注时，如果点错，按鼠标右键可以取消
标注完成后，如果不满意，可以点击绿色边框(边框会变红，如下图所示)，按Delete 删除

![](https://spire.imdo.co/images/2004/spire-tools-delete-label.jpg)

#### 将标注输出为 Yolo 格式，准备训练

在标注完成之后，按下 Ctrl+O
```



点击确定后



然后将下面 4 个文件取出用于 Yolo 训练

开始训练YOLO

使用YOLOv4和YOLOv3：

1. 针对选择的模型，下载预训练权重：
    * 对于 `yolov4.cfg`, `yolov4-custom.cfg` (162 MB): [yolov4.conv.137](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137)
    * 对于 `csresnext50-panet-spp.cfg` (133 MB): [csresnext50-panet-spp.conv.112](https://drive.google.com/file/d/16yMYCLQTY_oDlCIZPfn_sab6KD3zgzGq/view?usp=sharing)
    * 对于 `yolov3.cfg, yolov3-spp.cfg` (154 MB): [darknet53.conv.74](https://pjreddie.com/media/files/darknet53.conv.74)
    * 对于 `yolov3-tiny-prn.cfg , yolov3-tiny.cfg` (6 MB): [yolov3-tiny.conv.11](https://drive.google.com/file/d/18v36esoXCh-PsOKwyP2GWrpYDptDY8Zf/view?usp=sharing)
    * 对于 `enet-coco.cfg (EfficientNetB0-Yolov3)` (14 MB): [enetb0-coco.conv.132](https://drive.google.com/file/d/1uhh3D6RSn0ekgmsaTcl-ZW53WBaUDo6j/view?usp=sharing)

    > **百度网盘**打包下载，链接：https://pan.baidu.com/s/1CNVyyjoph7YVSXGT3vjbfQ，提取码：4usc

2. 将`cfg/yolov4-custom.cfg`拷贝一份，重命名为`yolov4-obj.cfg`（`obj`可以是自定义名称）
    * 修改batch为[`batch=64`](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L3)
    * 修改subdivisions为[`subdivisions=16`](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L4)
    * 修改max_batches为（`类别数量*2000`，但不要小于`4000`），如训练3个类别[`max_batches=6000`](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L20)
    * 修改steps为max_batches的0.8与0.9，如[`steps=4800,5400`](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L22)
    * 修改`classes=80`为自定义数据集的类别数量，主要需要修改3处（3个`[yolo]`层）：
        * 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
    * 修改`filters=255`为`filters=(classes+5)x3`，在3个`[yolo]`层的前一个`[convolutional]`层，分别为：
        * 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
    * 如果使用[`[Gaussian_yolo]`](https://github.com/AlexeyAB/darknet/blob/6e5bdf1282ad6b06ed0e962c3f5be67cf63d96dc/cfg/Gaussian_yolov3_BDD.cfg#L608)层，修改`filters=57`为`filters=(classes+9)x3`，在3个`[Gaussian_yolo]`层的前一个`

[convolutional]`层，分别为：
    * https://github.com/AlexeyAB/darknet/blob/6e5bdf1282ad6b06ed0e962c3f5be67cf63d96dc/cfg/Gaussian_yolov3_BDD.cfg#L604
    * https://github.com/AlexeyAB/darknet/blob/6e5bdf1282ad6b06ed0e962c3f5be67cf63d96dc/cfg/Gaussian_yolov3_BDD.cfg#L696
    * https://github.com/AlexeyAB/darknet/blob/6e5bdf1282ad6b06ed0e962c3f5be67cf63d96dc/cfg/Gaussian_yolov3_BDD.cfg#L789

> 例如，如果`classes=1`，则`filters=18`；如果`classes=2`，则`filters=21`。
> 注意：**不要**在`cfg`文件中直接写：`filters=(classes+5)x3)`

3. 在`darknet/data`路径下创建`obj.names`，其中每一行是一个目标类别名称

    * 将数据集标注得到的文件`Yolo_categories.names`重命名为`obj.names`，并放到`darknet/data`下

4. 在`darknet/data`路径下创建`obj.data`：

    * 教程 darknet 路径为 `/home/user/darknet`，本文以此为例，请根据自己的路径进行修改。在 /home/user/darknet/cfg/ 文件夹下新建一个文件，名字叫 obj.data 在里面写入：

    ```
    classes = 1
    train = /home/user/darknet/data/coco/Yolo_20180908_234114.txt
    valid = /home/user/darknet/data/coco/Yolo_20180908_234114.txt
    names = data/obj.names
    backup = backup
    eval = coco
    ```

    > 注意：classes 为类别数量，对于单类检测问题，写 1

5. 将图像文件（.jpg）与标注文件放入到如下路径`darknet\data\coco\`路径下

    * 将`scaled_images`里的图像拷贝到 `/home/user/darknet/data/coco/images/train`下
    * 将`Yolo_labels`里的标注文件拷贝到`/home/user/darknet/data/coco/images/train`下
    * 将`Yolo_20180908_234114.txt`拷贝到`/home/user/darknet/data/coco`下

6. 开始训练

    * 训练指令：`./darknet detector train data/obj.data cfg/yolo-obj.cfg yolov4.conv.137`
        * （对于最新100次迭代的最新权重`yolo-obj_last.weights`会保存在`darknet\backup\`）
        * （对于每1000次迭代的权重`yolo-obj_xxxx.weights`会保存在`darknet\backup\`）
        * （关闭Loss的显示窗口`./darknet detector train data/obj.data cfg/yolo-obj.cfg yolov4.conv.137 -dont_show`）
        * （通过浏览器查看训练过程`./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -dont_show -mjpeg_port 8090 -map`，然后打开Chrome浏览器，输入`http://ip-address:8090`）
        * （如果需要在训练中计算mAP，每4期计算一次，需要在`obj.data`文件中设置`valid=valid.txt`，运行：`./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -map`）

7. 训练结束，结果保存在`darknet\backup\yolo-obj_final.weights`
    * 如果训练中断，可以选择一个保存的权重继续训练，使用`./darknet detector train data/obj.data yolo-obj.cfg backup\yolo-obj_2000.weights`

> 注意：在训练中，如果`avg`（loss）出现`nan`，则训练出了问题，如果是其他字段出现`nan`，这

种情况是正常的。
> 注意：如果需要改变`cfg`文件中的`width=`或`height=`，新的数字需要被32整除。
> 注意：训练完成后，检测指令为：`./darknet detector test data/obj.data yolo-obj.cfg yolo-obj_8000.weights`。
> 注意：如果出现`Out of memory`，需要修改cfg文件中的`subdivisions=16`为`32`或`64`。

训练YOLOv3-Tiny

训练YOLOv3-Tiny与选了YOLOv4、YOLOv3基本相同，主要有以下小区别：

1. 下载yolov3-tiny预训练权重，运行命令`./darknet partial cfg/yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15`
2. 新建自定义`cfg`文件`yolov3-tiny-obj.cfg`（可以复制`cfg/yolov3-tiny.cfg`为`yolov3-tiny-obj.cfg`）
3. 运行训练命令：`./darknet detector train data/obj.data yolov3-tiny-obj.cfg yolov3-tiny.conv.15`

多GPU训练

1. 首先在1块GPU上训练1000次`./darknet detector train cfg/coco.data cfg/yolov4.cfg yolov4.conv.137`
2. 停止训练，使用权重`darknet/backup/yolov4_1000.weights`，在多块GPU上训练，运行`./darknet detector train cfg/coco.data cfg/yolov4.cfg /backup/yolov4_1000.weights -gpus 0,1,2,3`

> 注意：如果出现`nan`，应该降低学习率，如4块GPU`learning_rate=0.00065`（learning_rate=0.00261/GPUs），还应该增加cfg文件中的`burn_in=`为原先的4x，如`burn_in=4000`

训练常见程序问题

> 注意：如果出现如下错误

需要修改源码`/home/user/darknet/src/data.c`
将如下代码
```
list *get_paths(char *filename)
{
  char *path;
  FILE *file = fopen(filename, "r");
  if(!file)
    file_error(filename);
  list *lines = make_list();
  while((path=fgetl(file))) {
    list_insert(lines, path);
  }
  fclose(file);
  return lines;
}
```
修改为：
```
void ltrim(char *s)
{
  char *p; p = s;
```

```
  while (*p == ' ' || *p == '\t' || *p == '\r') { p++; } strcpy(s,p);
}

void rtrim(char *s)
{
  int i;
  i = strlen(s) - 1;
  while ((s[i] == ' ' || s[i] == '\t' || s[i] == '\r') && i >= 0 ) { i--; } s[i+1] =
'\0';
}

void _trim(char *s)
{
  ltrim(s);
  rtrim(s);
}

list *get_paths(char *filename)
{
  char *path;
  FILE *file = fopen(filename, "r"); if(!file) file_error(filename); list *lines =
make_list(); while((path=fgetl(file))) {
    _trim(path); list_insert(lines, path);
  }
  fclose(file); return lines;
}
```
保存，`make -j8`重新编译
下面为正常训练时画面

何时应该停止训练

通常情况下，为每个类别迭代2000次是足够的，且总的迭代次数不能低于4000次。但是如果想要更加精确的停止时间，可以参考以下说明：

1. 在训练过程中，你会看到一系列训练误差，当**0.XXXXXXX avg**这个参数不再下降时，就该停止训练了
   > 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
   > </br>
   > **9002**: 0.211667, **0.60730 avg**, 0.001000 rate, 3.868000 seconds, 576128 images Loaded: 0.000000 seconds

   * **9002** - 迭代数量（batch数量）
   * **0.60730 avg** - 平均损失（误差），**越低越好**

   如果发现**0.XXXXXXX avg**在很多次迭代后都不再降低，则是时候该停止训练了。最终的平均损失从0.05（对于小模型和简单训练数据）到3.0（对于大模型和复杂训练数据）不等。

2. 当训练停止之后，可以从`darknet\backup`中取出最新保存的训练权重`.weights`，并选择它们中检测效果最好的

   例如，当训练9000次停止后，效果最好的模型可能是之前保存权重中的一个（7000,8000,9000），这是因为过拟合（Overfiting）现象。过拟合的表现可以解释为，在训练图像上检测效果很好，但是在其他图像上效果不佳，这时候就该尽早停止训练（**早停点**）。

2.1 首先，你需要在`obj.data`中指定验证数据集`valid=valid.txt`，如果你没有准备验证数据集，可以简单的复制`data\train.txt`为`data\valid.txt`。

2.2 如果你在迭代9000次之后停止训练，验证之前的模型权重可以使用如下命令：

```
  *    ./darknet    detector    map    data/obj.data    cfg/yolo-obj.cfg    backup\yolo-obj_7000.weights
  *    ./darknet    detector    map    data/obj.data    cfg/yolo-obj.cfg    backup\yolo-obj_8000.weights
  *    ./darknet    detector    map    data/obj.data    cfg/yolo-obj.cfg    backup\yolo-obj_9000.weights
```

然后对比每个权重（7000,8000,9000）最后一行输出，选择**mAP（mean average precision）最高**权重，或者对比IoU（intersect over union）进行选择。

例如，**yolo-obj_8000.weights**的mAP最高，则使用这个权重。或者在训练时加上`-map`参数：

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

结果如下图所示，mAP每4期（Epoch）通过`obj.data`中设置的验证集`valid=valid.txt`上计算一次（1期=`train_txt中图像数量 / batch` 次迭代）。

运行训练好的模型，进行目标检测，执行：

```
./darknet detector test data/obj.data cfg/yolo-obj.cfg yolo-obj_8000.weights
```

如何提升检测效果

训练之前提升检测效果的技巧

* 设置`.cfg`文件中`random=1`，可以使用多分辨率输入增加检测效果：[link](https://github.com/AlexeyAB/darknet/blob/0039fd26786ab5f71d5af725fc18b3f521e7acfd/cfg/yolov3.cfg#L788)

* 在`.cfg`文件中增加网络的输入分辨率（设置任意可以被32整除的数字，如，`height=608`，`width=608`），可以增加精度

* 检查图像每个目标是否都被标记，图像中的所有目标都必须被正确标记，推荐使用数据管理工具检查：[spire-image-manager](https://github.com/jario-jin/spire-image-manager)

* Loss很大，mAP很低，是不是训练错了？在训练中使用`-show_imgs`参数，能够可视化目标框真值，检查数据集是否出了问题。

* 对于每一个你要检测的物体，在训练数据集中至少需要有一个实例与之相似，包括：形状、物体侧面、相对大小、旋转角度、倾斜方位角、光照等。因此，你的训练数据集需要包含具有不同对象属性的图像：比例、旋转、光照、不同侧面、不同背景等。建议对每一类物体收集2000张不同图像，并迭代训练**2000*类别数量**次。

* 推荐在训练数据集中包含带有不希望检测的非标记目标的图像。负样本图像不需要方框标记（空`.txt`文件），越多越好。

* 标注目标的最佳方式是：仅标注物体的可见部分，或标注物体的可见和重叠部分，或标注比整个物体稍多一点的部分（有一点间隙），标注你想让检测器检测的部分。

* 如果单幅图像中的物体很多，需要在`[yolo]`层或`[region]`层中修改参数`max=200`或者更高（全局最大目标检测数量为`0,0615234375*(width*height)`）。

* 如果想要检测小目标（图像被缩放到416×416后，小于16×16的目标）
  * 在https://github.com/AlexeyAB/darknet/blob/6f718c257815a984253346bba8fb7aa756c55090/cfg/yolov4.cfg#L895修改`layers = 23`
  * 在https://github.com/AlexeyAB/darknet/blob/6f718c257815a984253346bba8fb7aa756c55090/cfg/yolov4.cfg#L892修改`stride=4`
  * 在https://github.com/AlexeyAB/darknet/blob/6f718c257815a984253346bba8fb7aa756c55090/cfg/yolov4.cfg#L989修改`stride=4`

* 如果想要同时检测大目标与小目标，可以使用修改模型：
  * 全模型 - 5个yolo层：https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3_5l.cfg
  * 小模型 - 3个yolo层：https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3-tiny_3l.cfg
  * YOLOv4 - 3个yolo层：https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4-custom.cfg

* 如果你训练的数据类别需要区分左右目标（如检测左右手，交通信号中的左右方向），则不能使用左右翻转图像增强，在cfg文件中设置`flip=0`：https://github.com/AlexeyAB/darknet/blob/3d2d0a7c98dbc8923d9ff705b81ff4f7940ea6ff/cfg/yolov3.cfg#L17

* 一般规则 - 您的训练数据集应包含待检测目标的相对大小的集合：
  * `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`

  也就是，对于测试数据集中的每个物体，训练数据集中必须至少有一个具有相同类与大约相同相对大小的物体。如果训练数据中仅有占图像面积80-90%的物体，则训练后的网络不能够检测占图像面积1-10%的物体。

* 如果想加速训练（损失检测精度），可以在cfg文件layer-136中设置参数`stopbackward=1`

* 注意`物体的模型、侧面、光照、尺度、方位角`等属性，从神经网络的内部角度来看，这些是不同的物体。因此，你想检测的物体越多，就应该使用越复杂的网络模型。

* 如果想要外包矩形框更加精确，可以在`[yolo]`层中增加3个参数：`ignore_thresh=.9 iou_normalizer=0.5 iou_loss=giou`，这会增加mAP@0.9，同时降低mAP@0.5。

* 如果你比较熟悉检测网络了，可以重新计算自定义数据集的锚框（Anchor）：`./darknet detector calc_anchors data/obj.data -num_of_clusters 9 -width 416 -height 416`，然后设置cfg文件中3个`[yolo]`层9个锚框。同时需要改变每个`[yolo]`层中的锚框索引`mask=`，第一层有大于60×60的锚框，第二层有大于30×30的锚框，第三层相同。也需要改变每个`[yolo]`层之前的`filters=(classes + 5)*<number of mask>`。如果许多计算出的锚框不适合在适当的层下 - 那么就尝试使用默认锚框。

训练之后提升检测效果的技巧

* 增加cfg文件中网络输入的分辨率，如，`height=608`，`width=608`，或`height=832`，`width=832`，这样可以检测更小的目标。

TX2上的准备工作

* 推荐使用**Ubuntu 18.04**（可以使用[JetPack](https://developer.nvidia.com/embedded/

jetpack)刷机）
* **CMake >= 3.8**: https://cmake.org/download/
* **CUDA >= 10.0**: https://developer.nvidia.com/cuda-toolkit-archive
* **cuDNN >= 7.0 for CUDA >= 10.0** https://developer.nvidia.com/rdp/cudnn-archive
* **OpenCV >= 2.4**: https://opencv.org/releases.html
* **GCC**
* **ROS Melodic**: http://wiki.ros.org/melodic/Installation

使用JetPack为TX2安装CUDA与cuDNN

* 下载JetPack，地址：https://developer.nvidia.com/embedded/jetpack

* 进入 sdkmanager-[version].[build].deb 所在的路径，其中version和build代表相应各自的编号，安装Debian包：

```
sudo apt install ./sdkmanager-[version].[build].deb
```

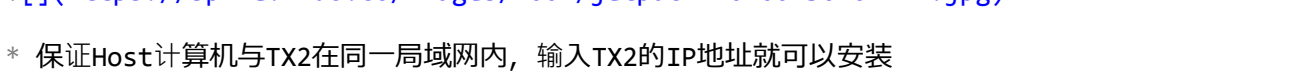
* 安装好之后，在Terminal中输入

```
sdkmanager
```

* 使用NVIDIA账号登录

* 选择开发环境
  * 在 Product Category 中选择 Jetson.
  * 在 Hardware Configuration 中选择 target hardware(Jetson TX2), **勾掉** host machine
  * 在 Target Operating System 中选择 JetPack 的版本.
  * 点击CONTINUE进入下一步

* 检查下载组件（如果仅安装CUDA和cuDNN，则只勾选红圈内的选项）、选择存储路径以及接收条款



* 保证Host计算机与TX2在同一局域网内，输入TX2的IP地址就可以安装

部署Darknet-ROS

* 下载darknet_ros源码

```
cd ~
cd catkin_ws/src
git clone --recursive https://github.com/leggedrobotics/darknet_ros.git
cd ../
```

* 编译

```
```

```
catkin_make -DCMAKE_BUILD_TYPE=Release
```

* 将训练好的cfg和weights加载到darknet_ros中

将`/home/user/darknet/cfg/yolov3-tiny.cfg`和`/home/user/darknet/backup`中刚刚训练好的参数
分别拷贝到`/home/user/catkin_ws/src/darknet_ros/darknet_ros/yolo_network_config`中的`cfg`和`weights`两个文件夹中
在`/home/user/catkin_ws/src/darknet_ros/darknet_ros/config`文件夹中新建`yolov3-tiny-obj.yaml`

里面写入

```
yolo_model:
  config_file:
    name: yolov3-tiny-obj.cfg
  weight_file:
    name: yolov3-tiny-obj.weights
  threshold:
    value: 0.3
  detection_classes:
    names:
      - drone
```

> 注意，在`yolov3-tiny-obj.yaml`文件中，需要指定刚才拷贝的`cfg`和`weights`文件以及`names`为自己训练的类别

在`/home/user/catkin_ws/src/darknet_ros/darknet_ros/launch`文件夹中，复制一份`darknet_ros.launch`，重命名为`obj_det.launch`
修改里面的
```
<rosparam command="load" ns="darknet_ros" file="$(find darknet_ros)/config/yolov2-tiny.yaml"/>
```
为
```
<rosparam command="load" ns="darknet_ros" file="$(find darknet_ros)/config/yolov3-tiny-obj.yaml"/>
```

> 注意：这正式刚才编写的yaml文件

roslaunch darknet_ros obj_det.launch

> 注意：进行检测，需要先打开一个ros_web_cam节点，以提供摄像头数据

最后，给一张YOLOv4检测结果的样张吧