Drone Detection Document

\*The main code was folked from <https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3>. The customed code is here: <https://github.com/Imogen1004/drone-detection>.

On the local machine, the code is stored in folder **work**.

# Configure the environment

First of all, we need to flash the Nvidia Jetson platform. There is a simple note about flashing the platform: <https://yuezhou896.medium.com/>. If we use Xavier, jut change the hardware in step 5.

After flashing, the latest jetpack will be installed (at my time the version is jetpack 4.5). OS, TensorRT, cuDNN, CUDA, OpenCV and some packages are included in jetpack.

In order to use our object detection model, we need to install python 3 first. Then install **TensorFlow 2.3.1**, we can follow this website: <https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html>.

Then we need to uninstall the previous version of OpenCV (4.1.1), which is incompatible with our application. We can install the proper version 4.5 by this bash file: <https://github.com/Imogen1004/JEP/blob/master/script/install_opencv4.5.0_Jetson.sh>.

It takes some time.

By running pip install -r ./requirements.txt in the terminal, we can configure or verify the version of other packages.

# Training dataset preperation

The annotated data should contain images and their corresponding xml file. The format of xml file should be like this:

<annotation>

<folder />

<filename>p1-frame-001.jpg</filename>

<path>/p1-frame-001.jpg</path>

<source>

<database>Unknown</database>

</source>

<size>

<width>1920</width>

<height>1080</height>

<depth>3</depth>

</size>

<segmented>0</segmented>

<object>

<name>phantom</name>

<pose>Unspecified</pose>

<truncated>Unspecified</truncated>

<difficult>0</difficult>

<bndbox>

<xmin>1290</xmin>

<ymin>407</ymin>

<xmax>1313</xmax>

<ymax>427</ymax>

</bndbox>

</object>

</annotation>

Images and their corresponding xml file should be put together in folder **custom\_dataset/train.** We should also separate 10% as test data and store them in **custom\_dataset/test.**

# Training a model

In yolov3->**configs.py**, choose the type of the model. We can apply transfer learning on four models: YOLOV3, YOLOV3 TINY, YOLOV4, YOLOV4 TINY.

The pre-trained weights should be stored in folder: **model\_data**, we can download weights we want by:

# yolov3

wget -P model\_data https://pjreddie.com/media/files/yolov3.weights

# yolov3-tiny

wget -P model\_data https://pjreddie.com/media/files/yolov3-tiny.weights

# yolov4

wget -P model\_data https://github.com/AlexeyAB/darknet/releases/download/darknet\_yolo\_v3\_optimal/yolov4.weights

# yolov4-tiny

wget -P model\_data <https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.weights>

Then we should change the following configuration:

YOLO\_TYPE ="yolov3" # yolov4 or yolov3

YOLO\_FRAMEWORK = "tf" # "tf" or "trt", when first training, using tf

TRAIN\_YOLO\_TINY = False #if we want tiny model, then true

TRAIN\_BATCH\_SIZE = 8

TRAIN\_EPOCHS = 30

Here, we can choose the model by configuring YOLO\_TYPE and TRAIN\_YOLO\_TINY, remember to set YOLO\_FRAMEWORK to tf if we train the model (trt is used when converting the model to tensorRT), we can change the TRAIN\_BATCH\_SIZE and TRAIN\_EPOCHS, I always set to 8 and 30.

After configuration, put the training dataset and testing dataset in folder: **custom\_dataset**, then run **tools/XML\_to\_YOLOv3.py** to generate the format suitable for training, then run **train.py** to train the model.

After training is finished (needs a longtime, if we have 5000 images, 1 hour per epoch), we can find our weights in folder: **checkpoints**. For example, the weights will be named as ‘yolov4\_custom.data-00000-of-00001’ and ‘yolov4\_custom.index’.

# Converting to TensorRT

First change the configuration:

YOLO\_TRT\_QUANTIZE\_MODE = "FP32" # INT8, FP16, FP32

YOLO\_CUSTOM\_WEIGHTS = "checkpoints/yolov4\_custom"

Then run **tools/ Convert\_to\_pb.py** and **tools/Convert\_to\_trt.py**.

(During this process, comment import cv2 in **yolov3/utils.py**, remember to uncomment it after the conversion is finished.)

After the first python file finished, a frozen model will appear in **checkpoints,** the name might be **yolov4-416** (depending on your configuration). After the second python file finished, a tensorRT model will appear in **checkpoints,** the name might be **yolov4-trt-FP32-416.**

Set configuration:

YOLO\_FRAMEWORK = "trt”

YOLO\_CUSTOM\_WEIGHTS = "checkpoints/yolov4\_trt-FP32-416"

Then we can use TensorRT model to do the detection.

# Using a model to do off-line detection

First choose the configuration. The main fields we need to configure:

YOLO\_TYPE = "yolov4" # yolov4 or yolov3

YOLO\_FRAMEWORK = "trt" # "tf" or "trt"

YOLO\_TRT\_QUANTIZE\_MODE = "FP32" # INT8, FP16, FP32

YOLO\_CUSTOM\_WEIGHTS = "checkpoints/yolov4\_custom"

TRAIN\_YOLO\_TINY = False

Fill in the weights folder in filed YOLO\_CUSTOM\_WEIGHTS, for example, ‘checkpoints/yolov4-416’, ‘checkpoints/yolov4-trt-FP32-416’.

Open **detection\_custom.py**, you can change the video\_path to your video. In the function **detect\_video**, you can set the output path to save the analyzed video (eg: ./IMAGES/output1.mp4), the score\_threshold can also be changed here, you can only show the boxes whose scores are higher than the score\_threshold.

Run **detection\_custom.py** to detect drones off-line**.**

# Using a model to do real-time detection

First choose the configuration. The main fields we need to configure:

YOLO\_TYPE = "yolov4" # yolov4 or yolov3

YOLO\_FRAMEWORK = "trt" # "tf" or "trt"

YOLO\_TRT\_QUANTIZE\_MODE = "FP32" # INT8, FP16, FP32

YOLO\_CUSTOM\_WEIGHTS = "checkpoints/yolov4\_custom"

TRAIN\_YOLO\_TINY = False

Fill in the weights folder in filed YOLO\_CUSTOM\_WEIGHTS, for example, ‘checkpoints/yolov4-416’, ‘checkpoints/yolov4-trt-FP32-416’.

Open **real-time.py**, we can change the video path by changing **cap=VideoCapture(“mypath”).** Then run **real-time.py**, the video data stream will be sent to the model and it will start to analyze each frame in the data stream.

When the model detects a drone, the information will be stored in a dictionary first, the dict will be written to a local json file, it will also be sent to the receiver (nodered), an image with bounding box will be stored locally too. The name of the local folder is **json\_detection**.