Object Tracking using YOLOv3, Deep Sort and Tensorflow.

This repository implements YOLOv3 and Deep SORT in order to perform real-time object tracking. Yolov3 is an algorithm that uses deep convolutional neural networks to perform object detection. We can feed these object detections into Deep SORT (Simple Online and Realtime Tracking with a Deep Association Metric) in order for a real-time object tracker to be created.

## Getting started

##### **Note:**

this is a general instruction documentation, you need to run all the commands below from a command prompt or a terminal being inside the folder of your choice implementation:

* for CPU: ~\yolov3\_deepsort\yolov3\_deepsort\_CPU
* for GPU: ~\yolov3\_deepsort\yolov3\_deepsort\_GPU

## The environment

To set up the environment you have two options: pip or conda, we recommend you to use conda if you are installing on your own pc, but it is easier to install with pip all the requirements if you are trying to run the tracker on google colaboratory.

## CPU implementation:

### Using Conda (Recommended if you are running this in your own pc)

#### # Tensorflow CPU

*conda env create -f conda-cpu.yml*

*conda activate tracker-cpu*

### Using Pip

#### # TensorFlow CPU

*pip install -r requirements.txt*

## GPU implementation:

#### Using Conda (Recommended if you are running this in your own pc)

#### # Tensorflow GPU

*conda env create -f conda-gpu.yml*

*conda activate tracker-gpu*

#### Using Pip

#### # TensorFlow GPU

*pip install -r requirements-gpu.txt*

### Nvidia Driver (For GPU, if you haven't set it up already)

#### # Ubuntu 18.04

*sudo add-apt-repository ppa:graphics-drivers/ppa*

*sudo apt install nvidia-driver-430# Windows/Other*

#### # Windows

*https://www.nvidia.com/Download/index.aspx*

## Downloading official pretrained weights

You can download the pretrained weights (and save them in ~/weights directory in case you are using windows), there are two different weight files you can use, a normal one focused on the accuracy of the detections and a tiny one, focused on the speed of the tracker. (you just need to select the one you want to use).

### For Linux: Let's download official yolov3 weights pretrained on COCO dataset.

#### # yolov3 (focused on accuracy)

*wget https://pjreddie.com/media/files/yolov3.weights -O weights/yolov3.weights*

#### # yolov3-tiny(focused on rapidness)

*wget https://pjreddie.com/media/files/yolov3-tiny.weights -O weights/yolov3-tiny.weights*

### For Windows: You can download the yolov3 weights by clicking [here](https://pjreddie.com/media/files/yolov3.weights) and yolov3-tiny [here](https://pjreddie.com/media/files/yolov3-tiny.weights) then save them to the weights folder.

## Using Custom trained weights

Learn How To Train Custom YOLOV3 Weights Here:<https://www.youtube.com/watch?v=zJDUhGL26iU>

Add your custom weights file to weights folder and your custom .names file into data/labels folder.

### 

### Saving your yolov3 weights as a TensorFlow model.

Load the weights using *load\_weights.py* script. This will convert the yolov3 weights into TensorFlow .tf model files!

# yolov3

*python load\_weights.py*

# yolov3-tiny

*python load\_weights.py --weights ./weights/yolov3-tiny.weights --output ./weights/yolov3-tiny.tf --tiny*

# yolov3-custom (add --tiny flag if your custom weights were trained for tiny model)

*python load\_weights.py --weights ./weights/<YOUR CUSTOM WEIGHTS FILE> --output ./weights/yolov3-custom.tf --num\_classes <# CLASSES>*

After executing one of the above lines, you should see proper .tf files in your weights folder. You are now ready to run object tracker.

## 

## 

## Running the Object Tracker

Now you can run the object tracker for whichever model you have created, pretrained, tiny, or custom.

# yolov3 on video

*python object\_tracker.py --video ./data/video/test.mp4 --output ./data/video/results.avi*

#yolov3 on webcam

*python object\_tracker.py --video 0 --output ./data/video/results.avi*

#yolov3-tiny

*python object\_tracker.py --video ./data/video/test.mp4 --output ./data/video/results.avi --weights ./weights/yolov3-tiny.tf --tiny*

#yolov3-custom (add --tiny flag if your custom weights were trained for tiny model)

*python object\_tracker.py --video ./data/video/test.mp4 --output ./data/video/results.avi --weights ./weights/yolov3-custom.tf --num\_classes <# CLASSES>*

## Important!

The output flag saves your object tracker results as an avi file for you to watch back. It is not necessary to have the flag if you don't want to save the resulting video.

There is a test video uploaded in the data/video folder called test.mp4. If you followed all the steps properly with the pretrained coco yolov3.weights model then when you run the object tracker with the first command above you should see the result video on the same folder as the original video.

NOTE: the object\_tracker.py script is going to create/update a “.json” file every time this is executed, the “.json” file will contain the frame, the track id, the center of the object detected and the bounding boxes of every detection made.

## Command Line Args Reference

### load\_weights.py:

*--output: path to output*

*(default: './weights/yolov3.tf')*

*--[no]tiny: yolov3 or yolov3-tiny*

*(default: 'false')*

*--weights: path to weights file*

*(default: './weights/yolov3.weights')*

*--num\_classes: number of classes in the model*

*(default: '80')*

*(an integer)*

### object\_tracker.py:

*--classes: path to classes file*

*(default: './data/labels/coco.names')*

*--video: path to input video (use 0 for webcam)*

*(default: './data/video/test.mp4')*

*--output: path to output video (remember to set right codec for given format. e.g. XVID for .avi)*

*(default: None)*

*--output\_format: codec used in VideoWriter when saving video to file*

*(default: 'XVID)*

*--[no]tiny: yolov3 or yolov3-tiny*

*(default: 'false')*

*--weights: path to weights file*

*(default: './weights/yolov3.tf')*

*--num\_classes: number of classes in the model*

*(default: '80')*

*(an integer)*

*--yolo\_max\_boxes: maximum number of detections at one time*

*(default: '100')*

*(an integer)*

*--yolo\_iou\_threshold: iou threshold for how close two boxes can be before they are detected as one box*

*(default: 0.5)*

*(a float)*

*--yolo\_score\_threshold: score threshold for confidence level in detection for detection to count*

*(default: 0.5)*

*(a float)*

## Acknowledgments

* [Yolov3 TensorFlow Amazing Implementation](https://github.com/zzh8829/yolov3-tf2)
* [Deep SORT Repository](https://github.com/nwojke/deep_sort)
* [Yolo v3 official paper](https://arxiv.org/abs/1804.02767)
* [Yolov3 and DeepSort implementation](https://www.youtube.com/watch?v=Cf1INvUsvkM)
* [Original README of the project](https://github.com/theAIGuysCode/yolov3_deepsort/blob/master/README.md)