Object Detection Tutorial (YOLO)

Description

In this tutorial we will go step by step on how to run state of the art object detection CNN (YOLO) using open source projects and TensorFlow, YOLO is a R-CNN network for detecting objects and proposing bounding boxes on them. It has more a lot of variations and configurations. We will focus on using the Tiny, 80 classes COCO one. YOLO is pretty lightweight and it achieves around 1FPS on Euclid. Impressive!

Link to the site and paper:

https://pjreddie.com/darknet/yolo/ **https://pjreddie.com/media/files/papers/yolo_1.pdf**

Requirements and setup

First of all we need to install a couple of packages and download some more from GitHub

1) TensorFlow (Install using pip)
2) DarkFlow – TensorFlow adaptation of Darknet network runner
3) Some configuration and weight files

1. Installing TensorFlow

Install TensorFlow using PIP : $ sudo pip install tensorflow

Note: if you don't have pip yet install it using: $ sudo apt-get install python-pip

2. Installing DarkFlow

DarkFlow is a network builder adapted from Darknet, it allows building TensorFlow networks from cfg files and loading pre trained weights. We will use it to run YOLO.

a. Clone DarkFlow from : https://github.com/thtrieu/darkflow
b. Weights and cfg files can be found on : https://pjreddie.com/darknet/yolo/
c. We will use Tiny-Yolo: COCO model. (Last One)
d. Download cfg and weights file and copy them to the **DarkFlow** folder
e. Classes names file coco.names (Found in the tutorial page)

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
Old YOLO	VOC 2007+2012	2007	63.4	40.19 Bn	45		link
SSD300	VOC 2007+2012	2007	74.3	-	46		link
SSD500	VOC 2007+2012	2007	76.8	-	19		link
YOLOv2	VOC 2007+2012	2007	76.8	34.90 Bn	67	cfg	weights
YOLOv2 544x544	VOC 2007+2012	2007	78.6	59.68 Bn	40	cfg	weights
Tiny YOLO	VOC 2007+2012	2007	57.1	6.97 Bn	207	cfg	weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200	cfg	weights

The Code

1. Yolo demo code over ROS using Euclid's cameras

```
#! /usr/bin/env python
import sys
sys.path.insert(0, '/opt/ros/kinetic/lib/python2.7/dist-packages/darkflow')

from net.build import TFNet
import cv2
import threading
import time
import rospy
import os
from cv_bridge import CvBridge
from sensor_msgs.msg import Image

class EuclidObjectRecognizer():
    def __init__(self):
        self.detectionImage = self.image = None
        self.lastTime = time.time()
        self.elapsedTime = 1
        self.boxes = []

        script_path = os.path.dirname(os.path.realpath(__file__))

        self.options = {"model": os.path.join(script_path, "tiny-coco.cfg"), "load":
os.path.join(
            script_path, "tiny-yolo.weights"), "threshold": 0.20, "config":
script_path}

        self.tfnet = TFNet(self.options)
        self.colors = self.tfnet.meta['colors']
        self.classesColorMap = {}

        # Start ROS
        rospy.init_node("object_recognizer", anonymous=True)
        self.bridge = CvBridge()
        self.imagePub = rospy.Publisher("/euclid/object/live", Image)

        self.imageSub = rospy.Subscriber(
            "/camera/color/image_raw", Image, self.newColorImage)

        self.rate = rospy.Rate(10)
```

```
        while self.image == None:
            self.rate.sleep()

        self.liveThread = threading.Thread(target=self.liveRecognitionThread)
        self.liveThread.start()
        self.mainThread()

    def newColorImage(self, imageMsg):
        self.image = cv2.cvtColor(self.bridge.imgmsg_to_cv2(imageMsg),
cv2.COLOR_RGB2BGR)

    def getClassColor(self, className):
        if className in self.classesColorMap:
            return self.classesColorMap[className]
        self.classesColorMap[className] = self.colors[len(self.classesColorMap) + 10]

    def mainThread(self):
        h, w, _ = self.image.shape
        while not rospy.is_shutdown():
            self.detectionImage = self.image.copy()
            for bbox in self.boxes:
                left, top, right, bot, label = bbox['topleft']['x'], bbox['topleft'][
                    'y'], bbox['bottomright']['x'], bbox['bottomright']['y'],
bbox['label']

                color = self.getClassColor(label)

                cv2.rectangle(self.detectionImage, (left, top),(right, bot), color, 3)
                cv2.putText(self.detectionImage, label, (left, top - 12),0, 2e-3 * h,
color, 1)

            self.imagePub.publish(self.bridge.cv2_to_imgmsg(self.detectionImage,
"bgr8"))

            self.rate.sleep()

    def liveRecognitionThread(self):
        print "Starting live recognition"
        while not rospy.is_shutdown():
            self.lastTime = time.time()
            self.boxes = self.tfnet.return_predict(self.image)
            self.elapsedTime = time.time() - self.lastTime
        print "Live recognition Stopped"
if __name__ == "__main__":
    try:
```

```
    recgonizer = EuclidObjectRecognizer()
except Exception as e:
    print e
    rospy.signal_shutdown()
```

2. Setup the configuration and weights

1. Download the weights file
2. Download the configuration file
3. Download the classes names file coco.names

Change **self.options** to point to the configuration and weight files you download.

Note: I recommend saving the weights and configuration files in the same directory as the script or to change the script_path variable to point to that directory.

Note: The classes names should be in the same directory as the configuration file.

3. Run the script

Now we are ready to run the script, The script subscribes to the RGB camera topic, while YOLO runs in a background thread predicting bounding boxes. The script also draws these boxes into an OpenCV image and publishes the result using a ROS Topic.

$ python yolo_demo.py

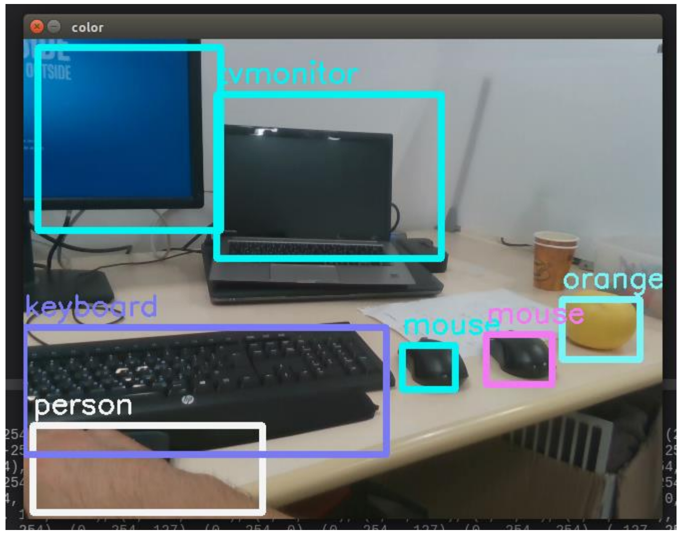
Note: Cameras node should be running in order to publish new images.

4. Time to play

Now you can try different weights and configurations, or simply start building an application using this amazing capability.

Some ideas:

f. Use depth to do tracking on the bounding boxes to get better results and FPS
g. Use the depth camera to calculate the distance of objects from Euclid
h. Use object detection for scene understanding and reasoning.


color
tvmonitor
keyboard
mouse
mouse
orange
person