ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

Real Time Monitoring System based on Yolo

Xiaoqin Liu[1], Ting Gong[1,*] and Yaqian Huang[1]

[1]Computer Science Department, North China Electric Power University (Baoding), Baoding China.

*Corresponding author Email: 1358130594@qq.com

Abstract

In this paper, aiming at the problem of object recognition in daily life, a real-time monitoring system based on Yolo is proposed. Because the real-time object is affected by many factors, the real-time object detection still has great challenges: (1) there are many interference factors in the real environment: rain, fog and other objective factors, target rotation, scaling and occlusion will have a lot of interference and influence on the detection, so how to eliminate the influence of environmental factors on the object detection process (2) How to make the detection speed of the detection system meet the real-time requirements, and how to improve the target detection speed (3) The demand of detection is increasing, for example, the type of the same target is different, and the model needs to distinguish it accurately. The detection system needs to obtain the category information and location information of the target object accurately at the same time. While ensuring the detection speed, the detection accuracy of the detection system becomes very important. Based on the above problems in real-time target detection scene, this paper proposes a unified real-time target detection model combined with the series of Yolo algorithms.

Keywords

Yolo; Real Time; Monitoring System.

1. Background and significance

When humans first invented computers, they began to think about how to make computers intelligent. Nowadays, artificial intelligence has become a very hot field, and has many active research topics and practical applications that benefit all aspects of life. At present, this field is growing exponentially, and will continue to develop healthily in the future. People hope to deal with some subjective and non-standard things automatically with the aid of artificial intelligence, such as recognition image and so on.

In the early stage of the development of artificial intelligence, it is easy for computer to deal with some problems that are difficult or even impossible for human beings to solve. These problems can be described by a formal mathematical law. The real tasks of artificial intelligence are those that are difficult to describe with formal symbols, which are easy to perform for human beings. For example, people can easily recognize what the other party says and the objects in the image. For this kind of problem, the computer can not give its own judgment.

In reality, things are extremely complex, it is difficult for people to go deep into things to see the essence, it is difficult to know which features are important, or even do not know what are the real features. The inspiration from the study of biological neural network is that we can let the machine explore the rules hidden in the knowledge independently, instead of simply instilling the knowledge into the computer, which will make the computer forget after learning like a naughty child. It is

ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

extremely difficult for human to extract highly abstract features from the original data, and the computer can express complex concepts with relatively simple models by simulating the human brain, which solves the key problem of feature extraction. Deep learning has gradually developed into an algorithm system with artificial neural network (ANN) algorithm as the core.

2. Introduction of related technologies

2.1 Keras

Keras is a model base, which provides a high-level building module for the development of deep learning model. It does not deal with low-level operations such as tensor product and convolution. On the contrary, it relies on a specialized and optimized tensor operation library to complete this operation, which can be used as the "back-end engine" of keras. Compared with choosing a tensor library alone and associating the implementation of keras with the library, keras deals with this problem in a modular way, and can seamlessly embed several different back-end engines into keras.

At present, keras can support three back ends: tensorflow back end, theano back end and cntk back end.

2.1.1 The module structure of keras

Back end: encapsulate tensorflow and theano to complete low-level tensor operation, calculation diagram compilation, etc

Model: model is an ordered combination of layers, a "container" of layers, and an overall representation of "neural network"

Layer: the layer of neural network essentially stipulates a calculation rule from input tensor to output tensor. Obviously, the whole neural network model is also such a calculation rule from tensor to tensor, so the model of keras is a subclass of layer.

The above three modules are the most important and core contents of keras. To build a neural network, just use the above contents.

2.2 Tensorflow

Tensorflow is an open source software library which uses data flow graph for numerical calculation. Nodes in the graph represent mathematical operations, while lines in the graph represent arrays, or tensors, that are related among nodes. Its flexible framework allows you to deploy computing on a variety of platforms, such as one or more CPUs (or GPUs) in desktop computers, servers, mobile devices, and so on. Tensorflow was originally developed by researchers and engineers of Google Machine Intelligence Research Institute for machine learning and deep neural network research. However, the versatility of this system makes it widely used in other computing fields.

2.3 OpenCv

Opencv is a cross platform computer vision library based on BSD license (open source), which can run on Linux, windows, Android and Mac OS operating systems. It is lightweight and efficient, which is composed of a series of C functions and a small number of C + + classes. It also provides the interfaces of python, ruby, MATLAB and other languages, and realizes many general algorithms in image processing and computer vision. Opencv is written in C + + language, its main interface is also c + + language, but it still retains a large number of C language interfaces.

The main application fields of OpenCV are: 1. Human-computer interaction 2. Object recognition 3. Image segmentation 4. Face recognition 5. Action recognition 6. Motion tracking 7. Robot 8. Motion analysis 9. Machine vision 10. Structure analysis 11. Safe driving.

3. Environment configuration and construction

The versions of environment configuration in this project are as follows:

python:3.6

ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

Anaconda: the latest edition

tensorflow:1.14.0

opencv:4.2.0.34

keras:2.1.5

4. Data set introduction and model building

4.1 Introduction to coco data set

Coco data set is a large and rich object detection, segmentation and subtitle data set. This data set takes scene understanding as the target, mainly intercepts from the complex daily scene, and the target in the image is calibrated by accurate segmentation. The images include 91 types of targets, 328000 images and 2500000 labels. So far, the largest dataset with semantic segmentation provides 80 categories, more than 330000 images, of which 200000 are labeled, and the number of individuals in the whole dataset is more than 1.5 million.

Format of coco dataset: Coco has five types of annotation: object detection, key point detection, instance segmentation, panoramic segmentation and image annotation, all of which correspond to a JSON file. JSON is a big dictionary, which contains the following keywords:

4.2 Model building

Step 1: open an empty folder named Yolo. Py, add the following code and import the package you need.

```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2
```

Figure 1. Pour in the desired bag

Step 2: we don't need image parameters, because here we are dealing with video streams and Videos - except that the following parameters remain unchanged:

– prototext: caffe prototext file path.

– model: the path of the pre training model.

– confidence: the minimum probability threshold for filtering weak detection. The default value is 20%.

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Figure 2. No modification required

Step 3: initialize the class list and color set. We initialize the class tag and the corresponding random colors.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

```
# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Figure 3. Initialize class tags and corresponding random colors

Step 4: load your own model and set your own video stream.

First, we load our own serialization model, and provide references to our own prototext file and model filenet = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"]).

Next, we initialize the video stream (the source can be a video file or a camera). First, we start videostream vs = videostream (SRC = 0). Start(), then wait for the camera to start time. Sleep (2.0), and finally calculate FPS = fps(). Start(). The videostream and FPS classes are part of the imutils package.

```
# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialize the video stream, allow the cammera sensor to warmup,
# and initialize the FPS counter
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
fps = FPS().start()
```

Figure 4. Initialize video stream

Step 5: traverse each frame

```
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # grab the frame from the threaded video file stream
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
        0.007843, (300, 300), 127.5)

    # pass the blob through the network and obtain the detections and
    # predictions
    net.setInput(blob)
    detections = net.forward()
```

Figure 5. Traverse each frame

First, read a frame frame = vs.read () from the video stream, then adjust its size imutils.resize (frame, width=400). Since we will need width and height later, we will grab (h, w) = frame. Shape [: 2]. Finally, the frame is transformed into a blob with DNN module, CV2. DNN. Blobfromimage (CV2. Resize (frame, (300, 300)), 0.007843, (300, 300), 127.5).

Now, let's set the blob as the input net. Setinput (BLOB) of the neural network, and pass the input detections = net. Forward () through net.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

Step 6: at this time, we have detected the target in the input frame. Now let's look at the confidence value to judge whether we can draw bounding boxes and labels around the target.

```
# loop over the detections
for i in np.arange(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with
    # the prediction
    confidence = detections[0, 0, i, 2]

    # filter out weak detections by ensuring the `confidence` is
    # greater than the minimum confidence
    if confidence > args["confidence"]:
        # extract the index of the class label from the
        # `detections`, then compute the (x, y)-coordinates of
        # the bounding box for the object
        idx = int(detections[0, 0, i, 1])
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        # draw the prediction on the frame
        label = "{}: {:.2f}%".format(CLASSES[idx],
            confidence * 100)
        cv2.rectangle(frame, (startX, startY), (endX, endY),
            COLORS[idx], 2)
        y = startY - 15 if startY - 15 > 15 else startY + 15
        cv2.putText(frame, label, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

Figure 6. Draw bounding boxes and labels around the target based on confidence

In the detection inner loop, multiple targets can be detected in one image. So we need to check the confidence. If the confidence level is high enough (higher than the threshold), the prediction will be displayed at the terminal, and the image will be predicted in the form of text and color bounding box.

In the inner loop of detections, first we extract the confidence value, confidence = detections [0, 0, I, 2]. If the confidence is higher than the minimum threshold (if confidence > args ["confidence]:), then extract the class Tag Index (IDX = int (detections [0, 0, I, 1])) and calculate the coordinates of the detected target (box = detections [0, 0, I, 3:7] * NP. Array ([w, h, W, H]). Then, we extract the (x, y) coordinates of the bounding box ((startx, starty, endx, Endy) = box. Astype ("int"), which will be used to draw rectangles and text. Next, a text label is constructed, which contains the class name and confidence (label = "{}: {. 2F}%". Format (classes [IDX], confidence * 100)). Also use the class color and the (x, y) coordinates extracted before to draw a color rectangle around the object (CV2. Rectangle (frame, (startx, starty), (endx, Endy), colors [IDX], 2)). If we want the label to appear above the rectangle, but if there is no space, we will display the label slightly below the top of the rectangle (y = starty - 15 if starty - 15 > 15 else starty + 15). Finally, we use the y value just calculated to put the color text on the frame (CV2. Puttext (frame, label, (startx, y), CV2. Font)_ HERSHEY_ SIMPLEX, 0.5, COLORS[idx], 2)).

The seventh step: frame capture cycle, the remaining steps also include: display frame; Check the quit key; Update the FPS counter.

```
# show the output frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop
if key == ord("q"):
    break

# update the FPS counter
fps.update()
```

Figure 7. Frame capture cycle

ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

The above code block is simple and clear. First, we show the frame (CV2. Imshow ("frame", frame)), then find the specific key (key = CV2. Waitkey (1) & 0xff), and check whether the "Q" key (representing "quit") is pressed. If it has been pressed, we exit the frame capture loop (if key = = ord ("Q"): break), and finally update the FPS counter (FPS. Update()).

Step 8: exit the loop (end of "Q" key or video stream), we have to deal with the following.

```
# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Figure 8. What needs to be handled in the launch cycle

5. Model testing

In the previous installation of Anaconda environment, enter the newly installed tensorflow environment, open the command window to test (in MAC, use Python to call the camera, due to limitations, you must use the command box to test), enter the Yolo project file location: CD / usr / local / keras-yolo3-ok. After entering the folder location, run Yolo_ Video.py file: Python Yolo_ video.py. In this way, the project will start successfully.

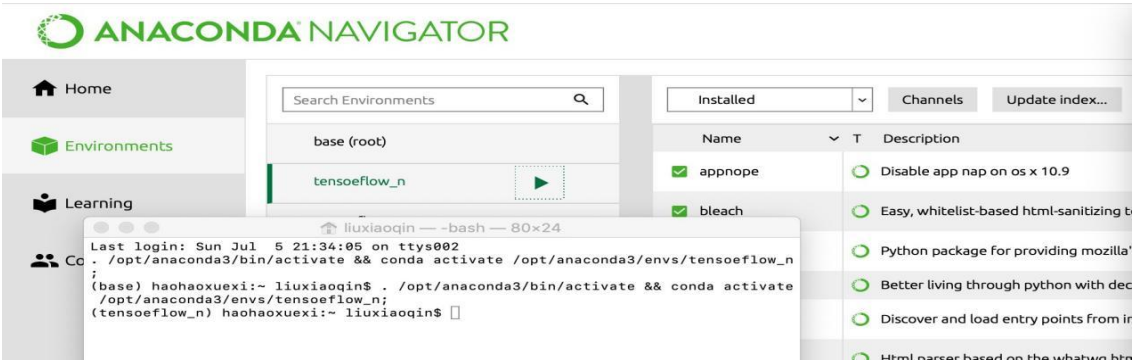

Figure 9. Launch project tutorial

The test results are as follows (here is a screenshot, the video is in the folder)

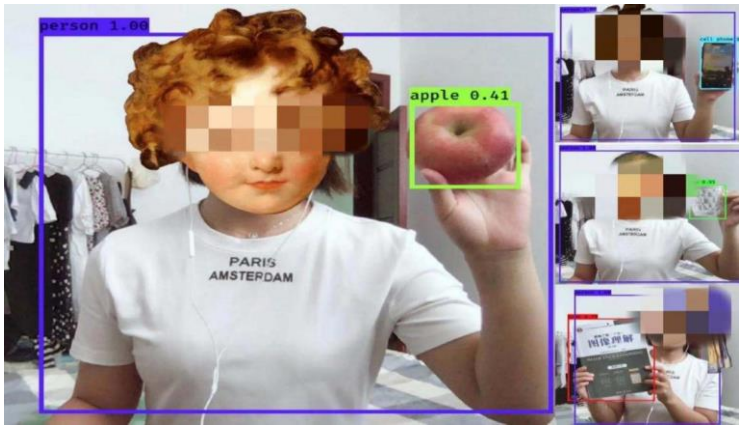

Figure 10. Display of test results

ISSN: 2414-1895 DOI: 10.6919/ICJE.202110_7(10).0064

6. Summary

The accuracy of the model is about 80%, and the confidence of each item is relatively high, which can identify most of the daily necessities. But there are still many problems, such as low recognition rate on untrained objects, slow speed of target detector and so on. All of these need to be improved. The tentative improvement method is: for the problem of low recognition rate of untrained items, we can use the method of increasing the types of training items to improve; For the problem of slow speed of target detector, we can use skip frame, different variants of mobilenet (faster, but lower accuracy), quantized variant of squeezenet (I think it should be faster, because its network footprint is smaller), fastyolo (less volume base) and other methods to improve.

Through this real-time monitoring system, we have a certain understanding of the basic usage of the Yolo model. We can try to optimize the Yolo model, so that the model can identify more types, improve the recognition rate and recognition rate.

References

[1] Raskar Punam Sunil,Shah Sanjeevani Kiran. REAL TIME OBJECT-BASED VIDEO FORGERY DETECTION USING YOLO (V2)[J]. Forensic Science International, 2021(prepublish):

[2] Shi Pengfei, Jiang Qigang, Shi Chao, Xi Jing, Tao Guofang, Zhang Sen, Zhang Zhenchao, Liu Bin, Gao Xin, Wu Qian. Oil Well Detection via Large-Scale and High-Resolution Remote Sensing Images Based on Improved YOLO v4 [J]. Remote Sensing,2021,13(16):

[3] Hu Jianming, Zhi Xiyang, Shi Tianjun, Zhang Wei, Cui Yang, Zhao Shenggang. PAG-YOLO: A Portable Attention-Guided YOLO Network for Small Ship Detection[J]. Remote Sensing,2021,13(16):

[4] Lv Bo, Zhang Nanfeng, Lin Xintao, Zhang Yanxi, Liang Tianfen, Gao Xiangdong. Surface Defects Detection of Car Door Seals Based on Improved YOLO V3[J]. Journal of Physics: Conference Series, 2021, 1986(1):

[5] Ryu SeongEun, Chung KyungYong. Detection Model of Occluded Object Based on YOLO Using Hard-Example Mining and Augmentation Policy Optimization[J]. Applied Sciences, 2021,11(15):

[6] Gu Jin, Ruo Suyun. Vehicle detection method based on improved YOLO v3 [J]. Agricultural Equipment and Vehicle Engineering, 2021, 59(07): 98-103.

[7] Shupeng Ji, Qisheng Wang, Shiyu Wu, Jiachen Tian, Xiao Li, Wenjin Wang. Deep Learning Based User Grouping for FD-MIMO Systems Exploiting Statistical Channel State Information [J]. China Communications, 2021, 18(07): 183-196.

[8] Liu Ting, Zhou Baijun, Zhao Yongsheng, Yan Shun. Ship Detection Algorithm based on Improved YOLO V5 [A]. Dalian Maritime University, Hong Kong Society of Mechanical Engineers (HKSME). Proceedings of 2021 6th International Conference on Automation, Control and Robotics Engineering (CACRE) [C]. Dalian Maritime University, Hong Kong Society of Mechanical Engineers (HKSME): Chengdu Sherlock Education Consulting Co., Ltd., 2021:5.

[9] Zhang Lanyong, Li Chengyu, Sun Hongfang. Object detection/tracking toward underwater photographs by remotely operated vehicles (ROVs) [J]. Future Generation Computer Systems, 2021(prepublish):

[10] Liu Tao, Pang Bo, Zhang Lei, Yang Wei, Sun Xiaoqiang. Sea Surface Object Detection Algorithm Based on YOLO v4 Fused with Reverse Depthwise Separable Convolution (RDSC) for USV[J]. Journal of Marine Science and Engineering,2021,9(7):

[11] Gao Hui, Wang Wenhao, Yang Chengjin, Jiao Weile, Chen Ziwei, Zhang Tong. Traffic signal image detection technology based on YOLO[J]. Journal of Physics: Conference Series, 2021, 1961(1):

[12] Isa Nur Ashiqin Mat,Mangshor Nur Nabilah Abu. Acne Type Recognition for Mobile-Based Application Using YOLO [J]. Journal of Physics: Conference Series,2021,1962(1):