YOLO Video Detection Project Report

Gary Yuan

Abstract

In this project, we build up a system where you could upload your own video and the system will analyze what kinds of objects are displayed in your video. Based on YOLOv3, the system is able to process images and videos in high speed and accuracy.

The system mainly adopted the MVC structure. It has a frontend page using html and css. In the model layer, we use flask to build up a server to process all the requests, and use YOLO and OpenCV to process the video files. In the model layer, we use Cassandra to store the data.

As the whole system has been completely dockerized, the users can conveniently use docker to download the system without annoying configurations.

Finally, after many tests, the system is verified to have basic video detection ability, and can clearly shows the users what kinds of objects have been detected in the video as the result can be played within the browser.

Key words: Docker; Flask; Cassandra; MVC; YOLO;

Introduction

In today's days when big data is so hot, how to use big data and store it is a very important topic. From this, we can imagine that a series of containers to build a series of microservices, in a series of distributed databases, to deal with these huge amounts of data.

In this project, we will borrow the traditional YOLO system, a object recognition system, to build a data processing system on it. We first borrow YOLO to process the video, the processed data is sent to the front end for display, while sending the data to the back end. Using a large distributed database for data storage, we finally packed it into containers and built a series of microservices.

The object detection system have wild applications, and by building it into microservices, the users can easily put it into use without difficult configurations. The report mainly introduces the working principle of the object detection system and how to implement it.

The Method

YOLO

You only look once (YOLO) is a state-of-the-art, real-time object detection system. YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model.

YOLO is famous for its real-time object detection function, however in this project we will only use it to process existed video files. Its speed and accuracy enable us to get the result in a relatively short time.

Docker

Docker is a set of coupled software-as-a-service and platform-as-a-service products that use operating-system-level virtualization to develop and deliver software in packages called containers.

In our system, we finally packed all our service into a docker container, so that the users can easily download and configure our program.

Flask

Flask is a lightweight WSGI web application framework. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

In our system, we use flask to build up the backend, send the html files, setup cassandra sessions, and send the video file for analysis and data storage.

Cassandra

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients.

In this project we use docker-cassandra to avoid annoying configurations. We put the set up of key space and table inside our main program, so that once you open our website, the key space and the table will be established.

Implementation

Building up the server

Flask is used in building up the server. Flask is a light weight web application framework, and thanks to its abundant functionality and handiness, we are able to build up the backend with ease.

The homepage will start with a simple html, and as long as the user opens this page, it will create a key space in cassandra.

```
@app.route('/')
def home():
    create_key_space()
    return render_template('index.html')
```

After having submitted the video file, the page will be directed to the upload page, where the video will be analyzed and the result will be shown within the browser.

```
@app.route('/upload', methods=['GET', 'POST'])
def get_video():
```

```
    if request.method == 'POST':
        video = request.files['videoSource']

        try:
            video.save(os.path.join(UPLOAD_PATH, video.filename))
        except Exception as e:
            print(e)


        detect_result = detect_object(video.filename)
        insert_data(video.filename, detect_result)
        return render_template('detect_result.html', result=detect_result,
filename=video.filename)
```

Processing the video

As YOLO has not supported the object detection for a whole video file, we have to splice the video file into a series of images using OpenCV. By analyzing the images, YOLO is able to detect the object with in the images, and gives reply about the location of the object in the image. Then OpenCV is utilized to draw rectangles around the objects, and give them tags around them. Finally, the images are put together into a new video using OpenCV.

The core code is shown below:

```
def detect_object(filename):
    net = cv2.dnn.readNet(WEIGHTS_DIR, CONFIG_DIR)
    classes = []
    print('start loading video ... ')

    fps = 16
    videowriter = None

    try:
        with open(NAMES_DIR, "r") as f:
            classes = [line.strip() for line in f.readlines()]
    except Exception as e:
        print(e)

    layer_names = net.getLayerNames()
    output_layers = [layer_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]
    colors = np.random.uniform(0, 255, size=(len(classes), 3))

    # Loading image
    print('loading image ...')
    file_dir = VIDEO_DIR + filename
    video = cv2.VideoCapture(file_dir)
    index = 1
    object_count = 0
    result = []
    flag_set_videowriter = 0

    ret, img = video.read()
    while (video.isOpened() and ret):
        # img = cv2.imread("room_ser.jpg")
```

```
        img = cv2.resize(img, None, fx=0.4, fy=0.4)
        height, width, channels = img.shape

        if flag_set_videowriter == 0:
            size = (width,height)
            try:
                videowriter =
cv2.VideoWriter(RELEASE_DIR+"result.mp4",cv2.VideoWriter_fourcc('H','2','6','4')
,fps,size)
            except Exception as e:
                print(e)
                print("failed to initiate videowriter")

            flag_set_videowriter = 1


        # Detecting objects
        blob = cv2.dnn.blobFromImage(img,
                                     0.00392, (416, 416), (0, 0, 0),
                                     True,
                                     crop=False)

        net.setInput(blob)
        outs = net.forward(output_layers)

        # Showing informations on the screen
        class_ids = []
        confidences = []
        boxes = []
        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > 0.5:
                    # Object detected
                    object_count += 1

                    center_x = int(detection[0] * width)
                    center_y = int(detection[1] * height)
                    w = int(detection[2] * width)
                    h = int(detection[3] * height)

                    # Rectangle coordinates
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)

                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

                    result.append([object_count, index, classes[class_id], x, y
,w, h])


        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
        # print(indexes)
        font = cv2.FONT_HERSHEY_PLAIN
```

```
        for i in range(len(boxes)):
            if i in indexes:
                x, y, w, h = boxes[i]
                label = str(classes[class_ids[i]])
                color = colors[i]
                cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
                cv2.putText(img, label, (x, y + 30), font, 1, color, 2)

        videowriter.write(img)

        index += 1
        ret, img = video.read()

    print('detect completed, returning ...')
    videowriter.release()
    return result
```

Storing the data

We choose to use Cassandra to implement the data persistence. As it's rather difficult to configure cassandra, we use docker cassandra to simplify the configuration. In the code, we use a python package -- cassadra-driver -- to interact with cassandra using python.

The set up of docker cassandra is done in the console, however the establishment of key space and tables are done in the python code. We do this to avoid useless tables and key spaces. As long as the user reach the home page, the key space and tables will be set up. The data about the objects in the video will also be stored in cassandra as long as YOLO finishes detection.

The core code is shown below:

```
def insert_data(filename, objects):
    try:
        cluster = Cluster(contact_points=['172.18.0.3'], port=9042)
        session = cluster.connect()
        print('connected to cassandra in inserting data')

    except Exception as e:
        print(e)
        print('connetion failed in inserting data...')

    session.set_keyspace(KEY_SPACE)

    print('Inserting data ...')
    try:
        for item in objects:
            session.execute('''
            INSERT INTO objects (filename, id, frame, class, x, y, w, h)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            ''',(filename, str(item[0]), str(item[1]), str(item[2]),
str(item[3]), str(item[4]), str(item[5]), str(item[6])))
    except Exception as e:
        print(e)

def create_key_space():
```

```
    try:
        # cluster = Cluster(contact_points=['0.0.0.0'], port=9042)
        cluster = Cluster(contact_points=['172.18.0.3'], port=9042)
        session = cluster.connect()
        print('connected to cassandra')

    except Exception as e:
        print(e)
        print('connetion failed ...')


    try:
        session.execute('''
        CREATE KEYSPACE IF NOT EXISTS %s
        WITH replication={'class':'SimpleStrategy', 'replication_factor' : 2}
        '''%KEY_SPACE)

        print('setting key space')
        session.set_keyspace(KEY_SPACE)

        print('creating table')
        session.execute('''
        CREATE TABLE  IF NOT EXISTS objects(
            filename text,
            id text,
            frame text,
            class text,
            x text,
            y text,
            w text,
            h text,
            PRIMARY KEY (filename, id)
        )
        ''')
        print('succeed')
    except Exception as e:
        print('create key space failed')
        print(e)
```

Result and Conclusion

An example of the video detection system is shown in this video.

The system is capable of reading the video file uploaded, saving it to its own space, calling the YOLO and OpenCV to analyze the objects in it, and finally writing the data into cassandra using python. The whole project has been dockerized, so the user doesn't have to do annoying configurations.

The system is easy and handy to use. All user has to do is to typing several commands in the console and upload one video file. And thanks to the fastness and accuracy provided by YOLO, the system can process the video file uploaded with fast speed, and output can clearly indicate the location and type of the objects detected in the video.

The system is not perfect, though. The front end webpage can be furtherly beautified, and show some prompts to the users when needed. Some special verifications can be added when coping with cassandra to ensure the data stored is absolutely corrected.

Through this project, I finally get familiar with the big data. The big data not only include the algorithms, but also include the processing, storing of the data, and how to provide service based on the data. Thus, in this project, I utilized YOLO to process the data, Cassandra to store the data, and docker to provide microservices based on the data.

At the same time, in this project I also get a lot of help from my teacher and classmates. By communicating with my classmates, we get through a lot of problems that all of us have encountered, and meanwhile make us think deeper about the big data. Also, my mentor gave me a lot of suggestions and tips on how to make my system better.