Real-Time Object Detection with Heterogeneous Models in a Distributed System

Long Nguyen
Computer Science
Penn State Harrisburg

***Abstract*—Object detection is one of the major areas in the computer vision field. This paper proposes a simple architecture for real-time object detection in a distributed system where multiple object detection models can be used in parallel. The proposed architecture can also be applied to tasks other than object detection, thus making it universally useful.**

***Index Terms*—Object Detection, Computer Vision, Distributed System**

I. Introduction

Real-Time video object detection has many useful applications. These applications include surveillance cameras, robots, autonomous driving cars, and object detection web services. Object detection refers to finding what objects are in the image, and where each object is. Object detection can take a long time to run with one machine, given a large enough object detection model. This implies if there are multiple object detection models running in one machine, it is not ideal for real-time applications. However, with a distributed system, the workload can be distributed to other machines, thus making the detection faster. The main objective of this project is to make object detection more efficient and faster when dealing with multiple object detection models. The object detection algorithm used in this project is *You Only Look Once (YOLO)* version 5 [1]. The entire application is written in Python since YOLOv5 is written in Python. The project is implemented with Python socket module [2], multithreading module [3], multiprocessing module [4], and some modifications to the YOLOv5 algorithm.

II. Literature review

A. You Only Look Once Algorithm

The *You Only Look Once (YOLO)* algorithm was first introduced by Redmon *et al.* [5]. The algorithm makes predictions based on Convolution Neural Network (CNN). As the name implies, the algorithm only needs to look at the image once before it makes predictions. First, the input image is evenly divided into regions. Each region is then passed to the CNN model, The CNN predicts the objects in the image and where they are, given the region of the image provided. After all regions of the image are passed to the CNN model, multiple objects are detected with *bounding boxes* (where they are). To eliminate the predictions with low probabilities, a confidence threshold is applied. After that, the algorithm uses a method called *Intersection over union*, to take care of the case where multiple bounding boxes overlap each other [5, 6].

B. Distributed Edge Cloud R-CNN

An attempt was made to make object detection possible under a distributed system. The first is from the paper *Distributed Edge Cloud R-CNN for Real Time Object Detection* [7]. This paper proposed a theoretical design of a new distributed algorithm called Distributed Region-based Convolutional Neural Network (RCNN). The proposed method tried to break the RCNN algorithm into different parts and distribute those parts across devices. The algorithm is not yet completed as the paper only showed one part of it done.

C. Multiple worker nodes running the same object detection model

Another attempt is from the paper *A Distributed Framework for Real Time Object Detection at Low Frame Rates with IoT Edge Nodes* [8]. For the overview of this approach, the system has two main types of nodes, one is the Head node, and the other is the Worker node. Worker nodes are responsible for processing the image. The Head node is responsible for taking in input from IoT camera device and assigning the work to the Worker nodes. The Head node sends the input video to each Worker node one frame at a time (no broadcast). Each Worker node then processes the frame it received from the Head node, then sends the results back to the Head node. The Head node saves the results received from the Worker nodes (it could do anything with the result, but the author chose to make it saves the results). After sending the result to the Head node, the Worker node then disconnects from the Head node and re-connects to the Head node, and is ready to process the next frame. This approach performs well and guarantees correct ordering for the processed frames. However, one limitation of this approach is that all Worker nodes have to use the same object detection model to process the input video. This means we cannot use multiple object detection models in our system concurrently.

From analyzing the two attempts at building a distributed system for object detection, we proposed a simple architecture for object detection in a distributed system that allows the concurrent use of heterogeneous models.

III. Methodology

A. Architecture overview

The architecture consists of four main components: *client*, *input server*, *worker servers*, and *output server*. The client

takes the input video and sends it frame by frame to the input server. The input server then broadcast each frame to all worker servers. Each worker server is responsible for running YOLOv5 algorithm to detect the objects in the frame. Then, the result is sent to the output server. The output server is responsible for receiving the results from all worker servers and combining them. Then, the output server sends the final results to the client.

Figure 1 shows the overview of our proposed architecture.

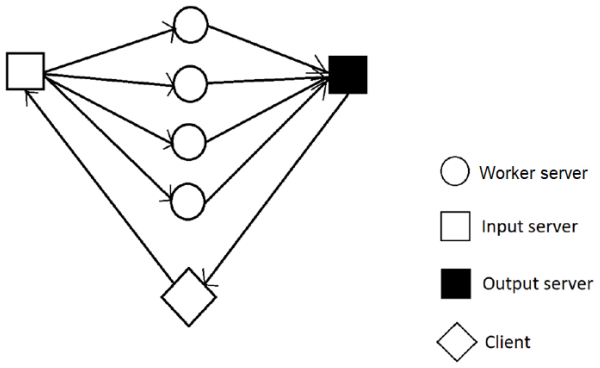

Fig. 1. A simple overview of the proposed architecture

B. Multiprocesses Synchronization (MS) Algorithm

The biggest challenge when building this architecture was the synchronization of broadcasting the input frame to all worker servers and the synchronization of the results received from the worker servers. We want to make sure all worker servers receive the input frame at the same time, and the results received from them are also received at the same time. The broadcasting happens at the input server and the receiving happens at the output server. We came up with an algorithm to overcome this challenge. The algorithm is called *Multiprocesses Synchronization (MS) Algorithm*, and it works as follows:

Shared Data
  `tasks_done` $\leftarrow multiprocessing.Manager.list()$
  `task` $\leftarrow multiprocessing.Manager.list()$

`coordinator``(tasks_done, task)`
  `tasks_done` $\leftarrow$ [**false**$*$n]
  {Update `task`}
  while true do
    if all (`tasks_done`) **then**
      {Update `task`}
      `tasks_done` $\leftarrow$ [**false**$*$n]
    end if
  end while

`handle_worker``(tasks_done, task, worker_id)`
  while true do
    if not `tasks_done[worker_id]` **then**
      {Do `task`}
      `tasks_done[worker_id]` $\leftarrow$ **true**
    end if
  end while

Note that n is the number of worker servers, and the shared data is process-safe and thread-safe. The above algorithm is correct. Each `handle_worker` has a `worker_id` which corresponds to an index of the shared list `tasks_done`. When all the `handle_worker` processes are done with their task, all elements in the `tasks_done` list will be **true**. When this happens, the `coordinator` will assign a new task and set every element in `tasks_done` to *false*. This ensures `handle_worker` processes can be run in parallel and be synchronized. Additionally, the variable `task` refers to any task, which implies that this algorithm can be used in other applications other than object detection.

C. Integrate YOLOv5 into the architecture

We used YOLOv5 algorithm in the worker servers. We modified the code to make sure we can call the detection function multiple times without the need to reconstruct the model. Each worker server is responsible for handling an object detection model. The result contained the objects detected with a confidence probability and the bounding box (points) for each object. After the detection is finished, the result is sent to the output server.

IV. Results

A. Dataset

To test our system, we trained three YOLOv5 models: one is for pet detection, one is for flower detection, and one is for vehicle types detection. The dataset for each model is taken from the Roboflow platform. The pet detection model can detect cats and dogs [9], the flower detection model can detect hortensia and malvaviscus flowers [10], and the vehicle types detection model can detect trucks and cars [11].

B. Environmental Setup

The setup to test our implementation of the architecture is as follows: the client ran on a separate machine, the input and output server ran on the same machine, and each worker server ran on each separate machine.

There are three workers, and each worker is responsible for handling an object detection model. As mentioned above, we trained three object detection models: pet detection, flower detection, and vehicle types detection. Each of them was trained on the pre-trained model "yolov5s.pt", which contained about 7.2 Million parameters. We trained each one with 50 epochs and 16 batches.

We tested our system on a local network, where each machine used (except for the one running the client) has the Intel(R) Xeon(R) W-2245 CPU @ 3.90GHz and the Quadro RTX 6000/8000 GPU. For the client, we ran it on a laptop

Fig. 2. Demo of synchronizing broadcast video frames from input server to worker servers

with an Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz and an integrated Intel(R) HD Graphics 620 GPU.

C. Synchronization

We integrated the MS algorithm into the input server and output server. For the input server, the task is to broadcast the input video frame by frame to all worker servers. For the output server, the task is the receive the results of each video frame from all worker servers.

Figure 2 shows the result of applying the algorithm to the input server, which broadcasts the input frame to all worker servers at the same time. In Figure 2, the worker servers display the received frame from the input server for demonstration purposes.

Similarly, the output server also received the results from all worker servers at the same time when the MS algorithm is applied.

D. Working system

The system worked as we expected it to. We made an input video with five objects in it: a cat, a dog, a hortensia flower, a truck, and a car. Figure 3 shows a screenshot of the input video we used. As we mentioned earlier, we trained three object detection models. When we ran the system, the worker that handled the pet detection model could detect the cat and the dog but nothing else. The worker that handled the flower detection model could detect the hortensia flower but nothing else. The worker that handled the vehicle types model could detect the truck and the car but nothing else.

However, the client was able to have all of those results thanks to the output server receiving them from all the worker servers. Figure 4 shows the final results from the client's perspective.

E. Client disconnected from and connected to a running system

When the client disconnected from a system that is running, the whole system waited for the new client to connect. Once the new client connected to the system, the system started its object detection task again.

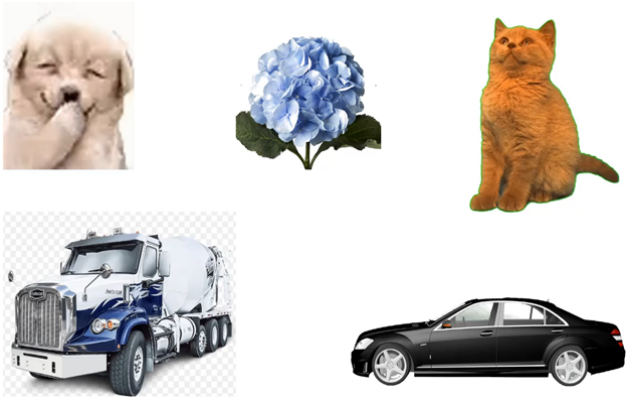

Fig. 3. Screenshot of the input video to test our system

Fig. 4. Client received all results from worker servers and displayed them

F. Worker server disconnected from and connected to a running system

When a worker server disconnected from a system that is running, the system simply kept running, but the objects detected by the disconnected server were no longer detected from the client's perspective. If no worker server is present, the system would wait for a new worker server to connect.

When a worker server is connected to a system that is running, it is automatically integrated into the system. The objects that could be detected by the new worker server are detected.

G. When the input or output server disconnected from a running system

When the input server or the output server is disconnected from the system, the whole system crash. This is because the proposed architecture is a centralized system and the input and output servers are the central components. If one of them crashes, the whole system crash.

V. Conclusion

This paper proposed a simple architecture for real-time object detection in a distributed system. This architecture allows the concurrent use of heterogeneous models. This architecture works best in a small network environment since the broadcasting of input frame to all worker servers is expensive in term of network traffics. This system will also save training time if we already have different types of models trained; we don't have to combine the datasets of the models that we want to use and make a new model that can detect it all.

Additionally, we came up with a new algorithm called Multiprocesses Synchronization (MS) Algorithm when building the architecture mentioned above. This algorithm is universally useful since it works with any kind of task and not just object detection tasks.

VI. Future Works

In the future, we plan to improve the fault tolerance of our system. If the input server or the output server crashed, the system will still keep going. We plan to use popular election algorithms for distributed systems such as Bully's Algorithm or Ring Election Algorithm to do that task. Furthermore, we plan to make our system capable of handling multiple clients by turning this architecture into a web service.

References

[1] G. Jocher, "ultralytics/yolov5," GitHub, Aug. 21, 2020. *https://github.com/ultralytics/yolov5*

[2] "socket — Low-level networking interface — Python 3.8.1 documentation," Python.org, 2020. *https://docs.python.org/3/library/socket.html*

[3] "threading — Thread-based parallelism — Python 3.9.0 documentation," docs.python.org. *https://docs.python.org/3/library/threading.html*

[4] "multiprocessing — Process-based parallelism — Python 3.8.3rc1 documentation," docs.python.org. *https://docs.python.org/3/library/multiprocessing.html*

[5] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[6] G. Karimi, "Introduction to YOLO Algorithm for Object Detection," Engineering Education (EngEd) Program — Section, Apr. 15, 2021. https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/

[7] J. Herrera, M. A. Demir, P. Yousefi, J. J. Prevost and P. Rad, "Distributed Edge Cloud R-CNN for Real Time Object Detection," 2018 World Automation Congress (WAC), Stevenson, WA, USA, 2018, pp. 1-5, doi: 10.23919/WAC.2018.8430385.

[8] L. K. Kalyanam, V. L. Ramnath, S. Katkoori and H. Zheng, "A Distributed Framework for Real Time Object Detection at Low Frame Rates with IoT Edge Nodes," 2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS), Chennai, India, 2020, pp. 285-290, doi: 10.1109/iSES50453.2020.00070.

[9] "Pet_Detection Object Detection Dataset by SreyoshiWorkSpace," Roboflow. *https://universe.roboflow.com/sreyoshiworkspace-radu9/pet_detection* (accessed May 04, 2023).

[10] "FLOWER1118 Object Detection Dataset and Pre-Trained Model by yolo," Roboflow. *https://universe.roboflow.com/yolo-c62gl/flower1118* (accessed May 04, 2023).

[11] "car models Object Detection Dataset and Pre-Trained Model by AIP," Roboflow. *https://universe.roboflow.com/aip-jbyra/car-models-u4taz* (accessed May 04, 2023).