Proceedings of the 5th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/29/20230708

Application of YOLOv5 for mask detection on IoT

Cheng Huang[1,4], Yishen Liu[2,5], Jiahao Li[1,6], Hao Tian[3,7], and Haoyi Chen[1,8]

[1]The Chinese University of Hong Kong, Hong Kong SAR, China
[2]Perking University, Shenzhen, China
[3]The Hong Kong Polytechnic University, Hong Kong SAR, China

[4]chenghuang@link.cuhk.edu.hk
[5]Lyshen2021@163.com
[6]leejarvis1314@gmail.com
[7]tianhao7214@gmail.com
[8]chenhaoyi0430@163.com

Abstract. The combination of the Internet of Things and deep learning technology is usually accompanied by many problems, such as limited bandwidth and computing resources. IoT combined with deep learning often causes system freezes or delays due to limited computing resources. Upgrading the hardware equipment of the IoT system requires a large economic cost, but using a lightweight deep learning model can reduce the consumption of hardware resources to adapt to the actual scene. In this paper, we combine IoT technology and improve a lightweight deep learning model, YOLOv5, to assist people in mask detection, vehicle counting, and target tracking, which does not take up too many computing resources. We deployed the improved YOLOv5 on the server side, and completed the training in the container. The weight file after training was deployed in Docker, and then combined with Kubernates to get the final experimental results. The resulting graph can be displayed by opening a browser at the edge node and entering the relevant IP address. Users can also perform certain operations on the results in the front end of the browser, such as drawing a horizontal line in the road to complete the local vehicle count. These operations are also fed back to the server for interaction with developers. For improved YOLOv5, the recognition speed and accuracy are faster than before. At the same time, compared with the previous version, the model itself requires less storage space and is easier to deploy, making the model easier to implement in the operation of edge nodes. Theoretical analysis and experimental results verify the feasibility and superiority of the proposed method.

Keywords: Internet of Things, deep learning, YOLOv5, object detection.

1. Introduction

With increase in speed and bandwidth of internet, Internet of Things (IoT) is pushing the market and the life of people to a new level and knocking on the door with new opportunities for interactions with other fields, such as 5G [1], artificial intelligence [2] and so on. For IoT itself, it is an extended network based on the Internet. It combines various information sensing devices with the network to form a huge network, so that people can operate and manage by issuing instructions to the devices of this network. For example, our home appliances can be connected to a server through IoT technology, and then users can use their mobile phones to command the server to control these furniture [3]. But with more and more devices adding IoT systems, IoT's central processing platform will face a large amount of data processing. How to deal with these huge amounts of data, or how to achieve efficient scheduling in limited computing resources to prevent congestion, has become a hot research direction. This means that running such a large amount of data requires powerful hardware. At the same time, better strategies must be developed to prevent conflict. But in many real cases, we often get limited resources, such as no powerful GPU and CPU, insufficient human resources, and imperfect allocation strategies, so the existing IoT systems will perform unsatisfactorily. For example, when researchers use cameras to count vehicles, a single node can run and compute without overloading the core server. However, if multiple edge nodes start running, transmitting video, etc., the server may be overloaded. At the same time, human resources will also increase to ensure and maintain the normal operation of the system. Based on such situations, developing a lightweight and easy-to-deploy 'tool' to assist people in serving IoT systems can not only reduce human resource consumption, but also help the system process large amounts of data more efficiently.

To make IoT application be used in real-world scenarios, many researchers combine artificial intelligence technology with IoT [4] [5] [6]. Artificial intelligence can operate and maintain the IoT system like a human. For example, it can help the communication of the IoT system to allocate bandwidth reasonably, and it can also complete some monitoring on the system: vehicle counting, mask detection, etc. Different tasks often require different artificial intelligence models, which also involves detailed branches of AI. For AI itself, there are many branches, such as machine learning, deep learning, reinforcement learning, etc. Many of these branches can also intersect with each other, such as deep reinforcement learning. Deep learning is to learn the inherent laws and representation levels of sample data, and the information obtained during the learning process is of great help to the interpretation of the data. Its ultimate goal is to enable machines to have the ability to analyze and learn like humans, and to recognize data such as words, images, and sounds, like image recognition [7] [8] and segmentation [9] [10], speech recognition [11] [12], target tracking [13] [14], etc.. This technologies can be used in some special application scenarios, such as during the new crown epidemic, people need to wear masks. Identifying whether people are wearing masks can help the government implement effective epidemic prevention measures. Epidemic prevention personnel can capture pictures of people by calling the cameras in the IoT system, and then use AI for mask identification.

In this paper, we combine the YOLOv5 model [15] with BiFPN [16] and Ghost-Net [17] to propose the improved YOLOv5, a lightweight model ,that can be used for dense object detection. This model is lightweight, which means that it does not require as much computational resources as larger models. Such a model does not preempt too many IoT system resources to keep itself running. This ensures that the system is not easily overloaded if multiple edge nodes start computing at the same time. At the same time, we combined Ghost-Net [17] to further reduce the redundant feature calculation in the image, which can be faster in speed. This can enhance the realtime performance of the system, reduce delays, and collect the latest mask wearing data at the first time. Finally, we introduce BiFPN [16] to replace the PANs network, in order to extract features bidirectionally. Doing so allows our model to have better detection performance than the original model when dealing with dense targets and small targets. On the basis again, we deploy this model on the IoT platform system, as shown in Fig. 1, which can effectively assist people in operations such as vehicle counting and mask detection. We collect videos from Pod Tracking and save it in Share Volume Store Videos. Our model is stored in Pod opendatacam. mongoDB is a database based on distributed file storage. The Pod RTSP Server is a real-time streaming protocol-based server, and its purpose is to transmit the processed data to the client. Pod HTTP Server is bound to an IP address and port number and listens for incoming TCP connections from clients on this address. Finally, you can open the web according to the corresponding port number to view the results.

The main contributions of this article are summarized as follows.

• We use the Ghost convolution to replace the convolution in YOLOv5 and modify the PANet layer in YOLOv5 to EfficientDet-BiFPN, which can make the model become a lightweight one, and realize the bidirectional fusion of topdown and bottom-up deep and shallow features, and enhance the transfer of feature information between different network layers.

• We combine the improved YOLOv5 model with IoT technology to make the system intelligent, which can assist in people counting and data collection in the real life situation. The IoT system can be easily deployed on the terminal of the server, and people can observe it by opening a specific web.

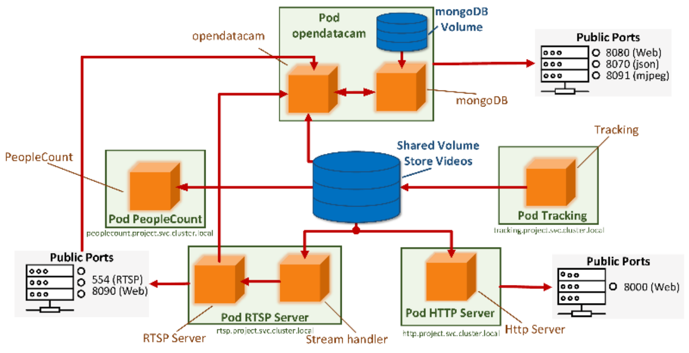


Figure 1. Overview diagram of our approach.

2. Methodology

2.1. IoT system

The establishment of the entire IoT system and how to deploy and operate on the edge node and Kubemetes platform are shown in Fig. 2. We combine the Dockerfile with the dependency file to create a docker file, and then upload the file to the docker hub. Furthermore, we store the YOLOv5 configuration file and the improved configuration file in config.json, and participate in the development of Kubemetes [18] together with docker hub through configmap. Then run it in opendatacampod, and apply for video data from mongoDB pod through port 27017. Finally, we can open a browser through port: 8070, 8080, or 8090 to view the output results.

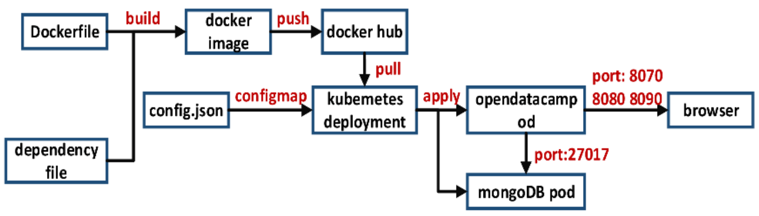


Figure 2. Flowchart of the IoT system we use.

And then, as shown in Fig. 1, this is the mapping of the system at the hardware level. We collect videos from Pod Tracking and save it in Share Volume Store Videos. Our model is stored in Pod opendatacam. mongoDB is a database based on distributed file storage. The Pod RTSP Server is a real-time streaming protocol-based server, and its purpose is to transmit the processed data to the client. Pod HTTP Server is bound to an IP address and port number and listens for incoming TCP connections from clients on this address. Finally, you can open the web according to the corresponding port number to view the results.

2.2. *YOLOv5 modification*

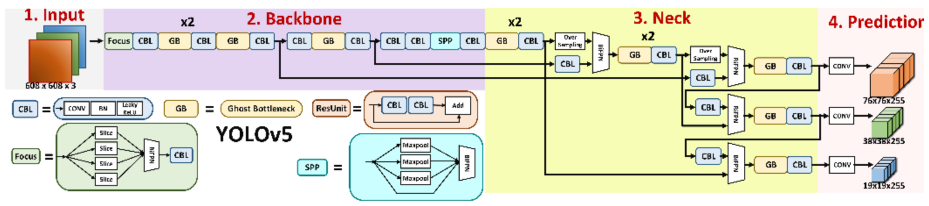


Figure 3. The structure of our proposed method.

As shown in Fig. 3 and Fig. 4, the general structure of the improved YOLOv5 algorithm has not changed. We replaced the original CSP structure with Ghost Bottleneck, which can greatly optimize the parameter scale and computing resource consumption of the network without affecting the detection accuracy of network pre-training. There are many other ways to reduce redundant parameter computation and redundant features, such as atrous convolution [19]. The receptive field of atrous convolution is very large. When the amount of parameters is certain, ordinary convolution can only extract small blocks of features, while atrous convolution can increase the hole rate to make more overlapping sampling areas on the input feature map for each sampling, so as to obtain denser characteristic response. Atrous convolution can be used when the network layer needs a large receptive field, but the computing resources are limited and the number or size of the convolution kernel cannot be increased. But this also brings two problems: not all pixels participate in the calculation, so the obtained features are discontinuous, which is defective in pixel-level detection; the information using a large dilation rate is only effective for some large object segmentation, while for small objects it does not help. When our model detects masks, some of the masks have a very small proportion of the screen, or the video is blurry and the resolution is not high. These situations will lead to difficulties in sampling and thus cannot be accurately identified. Therefore, atrous convolution is not suitable for this scenario.

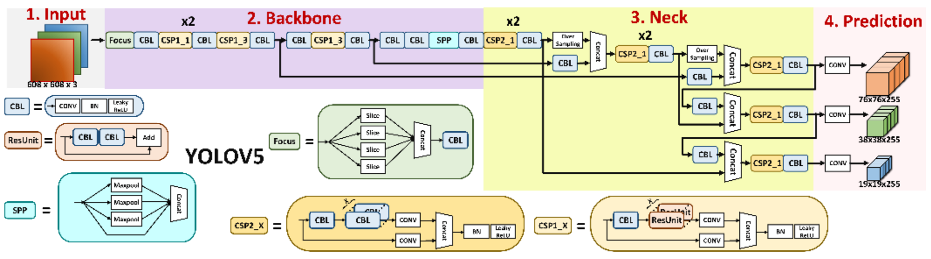


Figure 4. The structure of YOLOv5.

As shown in Fig. 5, the Module in GhostNet is divided into two steps to obtain the same number of feature maps as ordinary convolutions: i. smaller amount of convolution. ii. cheap operations. In this way, the feature maps of the upper and lower parts are phantoms of each other. The first step is to apply a traditional convolution operation to the input tensor to obtain a tensor of yellow feature maps from light to dark in the Fig. 3, also known as intrinsic features. Then, the linear operation represented by the ϕ function (Depthwise Separable Convolution) [20] [21] is applied to the feature map to generate a red feature map tensor from light to dark in the figure, and finally stacked together as the output of the Ghost Module, which is uesed to generate a corresponding Ghost feature map for a series of single-channel feature maps.

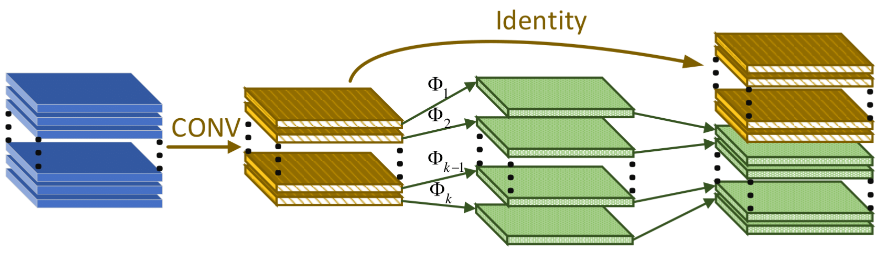


Figure 5. The Ghost module.

Set the width of the input tensor to be w, the height to be h, the number of channels to be c, the number of channels of the output tensor to be n, and the size of the standard convolution kernel to be k*k. The conv here should not contain nonlinear activation functions. The output of feature maps in the Ghost Module can be divided into n/s groups, each of which is s feature maps. The s feature maps contain the calculation result of a standard convolution kernel and the result obtained by s-1 linear transformations. The operation of adding offsets is ignored, and the approximation task k*k is equal to d*d. Then the ratio of computational complexity between ordinary convolution and Ghost-Net is as follows:

$$r_s = \frac{n \cdot h' \cdot w' \cdot c \cdot k \cdot k}{\frac{n}{s} \cdot h' \cdot w' \cdot c \cdot k \cdot k + (s-1) \cdot \frac{n}{s} \cdot h' \cdot w' \cdot d \cdot d} = \frac{c \cdot k \cdot k}{\frac{1}{s} \cdot c \cdot k \cdot k + \frac{s-1}{s} \cdot d \cdot d} \approx \frac{s \cdot c}{s + c - 1} \approx s \tag{1}$$

As shown in (1), using the Ghost Module to transform the standard convolution kernel can reduce the calculation amount and the number of parameters of the model by approximately s times.

Based on advantages of Ghost Modules, the author introduces the Ghost bottleneck (G-bneck) specially designed for small CNNs, as shown in Fig. 6. Borrowing from MobileNetV2, ReLU is not used after the second Ghost module, and other layers apply BN and ReLU after each layer. For efficiency reasons, the initial convolution in the Ghost module is a point convolution.

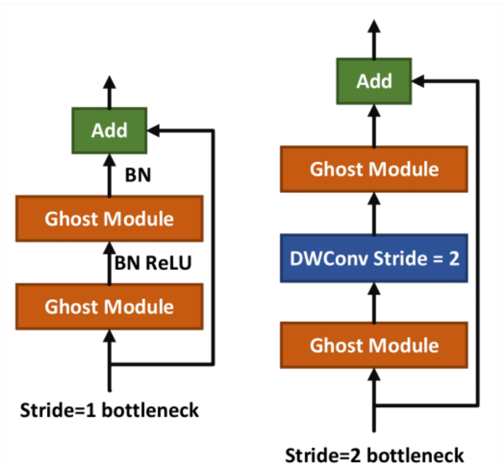


Figure 6. Ghost bottleneck with different parameters.

And this structure mainly adopts the common design pattern of channel enlargement and reduction. It first passes through a Ghost Module that enlarges the number of channels, and then connects to a Ghost module that reduces the number of channels, so that the number of channels before and after remains unchanged, so as to perform a channel-by-channel addition operation with the original tensor connected through the shortcut. There are two types of bottlenecks of this type. The structure on the right side in Fig. 6 is based on the structure on the left, and a Depthwise Conv with stride=2 is added.

Correspondingly, in order to be able to add directly, the shortcut branch needs to perform a downsampling.

And also, we replaced the Concat unit with BiFPN, which can realize the two-way fusion of top-down and bottomup deep and shallow features, and enhance the transfer of feature information between different network layers. Because the resolutions of the feature maps of different layers are different, their contributions to the fused features are different, so weights are introduced in the feature fusion stage. Unlike PANs, PANs adds a bottom-up channel based on FPN, which makes PANs have better accuracy than FPN, but requires more parameters and computation. If the node with only one input edge and output edge in the PANs is removed, and if the input and output nodes are at the same level, an extra edge is added to fuse more features without increasing the cost. As shown in (2), there is no limit on weight, which may cause unstable training. In (2), the calculation of softmax is slow, which is especially prominent in the front end. In order to ensure that the weight is greater than 0, the relu function is used before the weight. In order to ensure that the weight is greater than 0, the relu function is used before the weight. (3) Similar accuracy to (2) but can be 30% faster. Based on it, BiFPN achieves similar accuracy to repeated FPN+PANet, but uses far fewer parameters and FLOPs. With additional weighted feature fusion, our BiFPN further achieves the best accuracy with fewer parameters and FLOPs.

2.3. BiFPN

PANs [22] is an improved FPN algorithm [23] which adds a bottom-up channel, but it brings a problem, that is, a large amount of computation, and BiFPN [16] improves it on the basis of FPN, adds edges with contextual information to the original FPN module, and multiplies each edge with a corresponding weight. As shown in Fig. 7, compared with PANs, BiFPN adds residual links to enhance the representation ability of features; moreover, it removes nodes with a single input edge, because the nodes on the input edge do not perform feature fusion, so they have information comparison Less, there is no contribution to the final fusion, on the contrary, removal can also reduce the amount of calculation. At the same time, it also performs weight fusion, that is, adding a weight to each scale feature of the fusion to adjust the contribution of each scale. Among them, Fast-softmax is proposed to improve the detection speed.

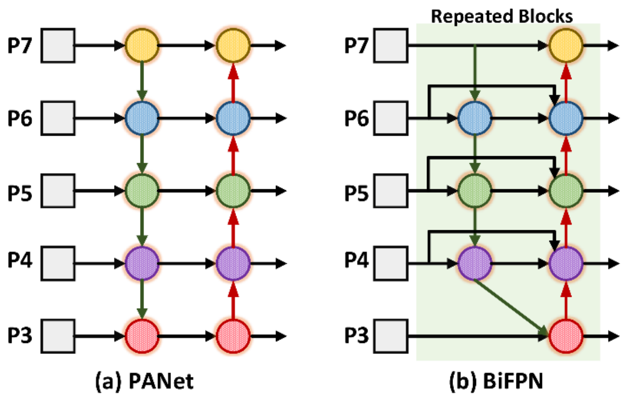


Figure 7. Structural comparison of PANs and BiFPN.

Since different input features have different resolutions, their contributions to output features are usually unequal. So let each input add extra weight and let the network understand the importance of each input feature. M. Tan and his team have considered three ways to solve the problem: i. Unbounded fusion; ii. Fusion based on softmax; iii. Fast normalized fusion.

As shown in (2), wi is a learnable weight that can represent vectors, scalars and multidimensional tensors. However, since the scalar weights are unbounded, the training will be unstable. Based on this, weight normalization is used to limit the value range.

$$O = \sum_i w_i \cdot I_i \tag{2}$$

Also, as shown in (3), (2) is combined with softmax yields (3).

$$O = \sum_i \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot I_i \tag{3}$$

(3) Apply softmax to each weight such that all weights are normalized to the 0 to 1 range to represent their input importance. And then, (3) adopts a fast fusion method, which is (4), and its value is passed through the Relu function to ensure numerical stability, and the value of its normalized weight is also between 0 and 1.

$$O = \sum_i \frac{w_j}{\epsilon + \sum_j w_j} \cdot I_i \tag{4}$$

In general, Bi-FPN is equivalent to giving different weights to each layer for fusion, making the network pay more attention to important layers, and reducing the node connections of some unnecessary layers.

3. Hardware equipment and data preparation

3.1. BiFPN

Our model and deployment of the iot system use the Ubuntu18.04 operating system, 16GB of memory, and a weak GPU acceleration. Several related servers are on this system.

The Docker weight file and the Kubernetes deployment are both on the personal computer. The final viewing of the results via the web was also performed on a personal computer.

3.2. Dataset

Our dataset comes from RMFD(Real-World Masked Face Dataset) [24]. The dataset is mainly divided into two parts: real face mask data and simulated mask face data. Among them, the real mask face recognition dataset contains 5,000 mask faces and 90,000 normal faces of 525 people. The simulated mask face recognition dataset includes a simulated mask face dataset of 10,000 people and 500,000 faces. These images can be annotated with Labelme software to generate labels, and then divide the dataset. We use a ratio of 8:2 to divide the training and testing datasets in this article.

3.3. Performance measurements

For the measurement of detection or recognition performance of machine learning model results, confusion metric is widely used, it is a performance-based metric, and the widely used metrics for model evaluation are discussed as follows:

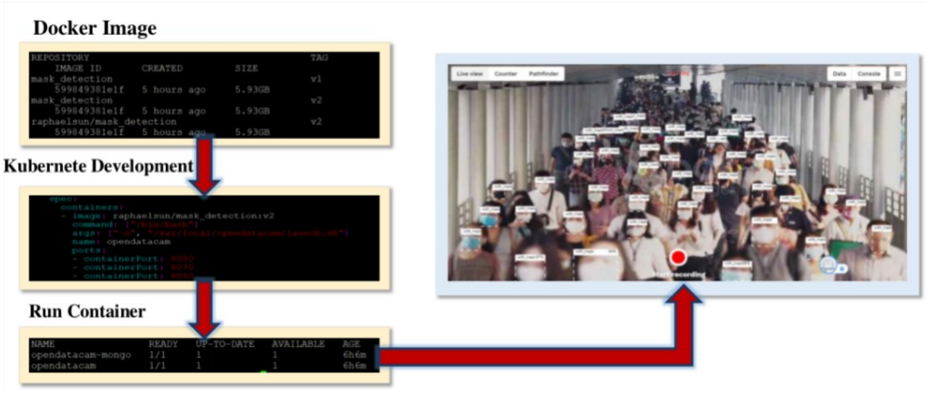
1. True Positive (TP): In attack detection, the TP indicates that Class A is correctly identified as belonging to Class A.
2. True Negative (TN): This matrix indicates that Class A is correctly identified as not belonging to Class A.
3. False Positive (FP): It indicates that Class A is not correctly identified as belonging to Class A.
4. False Negative (FN): It indicates that Class A is not correctly identified as not belonging to Class A.

However, using the metrics above, different measures can be made to better evaluate the model. For accurate detection, the classifier minimizes the values of the FP and FN metrics. However, the selected metrics used in this article are detailed below.

Accuracy: In mask detection, it can be described as Whether or not wearing a mask is a false detection that does not match the real situation. However, using performance measurement metrics, the accuracy can be defined mathematically as

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5}$$

And we only used the above given equation for the proposed technique performance evaluation.



```
REPOSITORY                                          TAG
    IMAGE ID        CREATED         SIZE
mask_detection                                      v1
    599849381e1f    5 hours ago     5.93GB
mask_detection                                      v2
    599849381e1f    5 hours ago     5.93GB
raphaelsun/mask_detection                           v2
    599849381e1f    5 hours ago     5.93GB
```

```
  spec:
    containers:
    - image: raphaelsun/mask_detection:v2
      command: ["/bin/bash"]
      args: ["-c", "/var/local/opendatacam/launch.sh"]
      name: opendatacam
      ports:
      - containerPort: 8080
      - containerPort: 8070
      - containerPort: 8090
```

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
opendatacam-mongo    1/1     1            1           6h6m
opendatacam          1/1     1            1           6h6m
```

Figure 8. The experimental result of our method.

4. Experimental results and analysis

4.1. Experimental results

As shown in Fig. 8, this is the final result of our experiment after combining the IoT system with the improved YOLOv5 model. We opened the IP address and its port output by the server on the personal computer, and the results showed the real-time detection of masks. The experimental results will not be displayed at the beginning, because the CPU is used for computing, and it will take a while to appear.

Table 1. Comparison of mask detection effects of different models.

Model	Mask	Unmask	FPS
YOLOv4	95.12%	95.12%	74
YOLOv5	96.62%	97.87%	107
Faster R-CNN	95.78%	95.31%	71
SSD	95.98%	96.63%	96
GB-YOLO	97.08%	97.85%	128

We first deployed the relevant Docker Image on the server, then combined it with Kubernetes, then we ran the container through Kubernetes, and finally opened the corresponding port through the web to see the result.

At the same time, we also our model separately, as shown in Fig. 9 and compared it with other models, as shown in Table. I. We compare v4, v5, SSD, Faster R-CNN and our model. These models are all run on the original operating system, and the same dataset is used for experiments. Among them, the test of the number of frames is mainly based on the video of mask detection, and the rest of the video such as vehicle counting is not included in it.

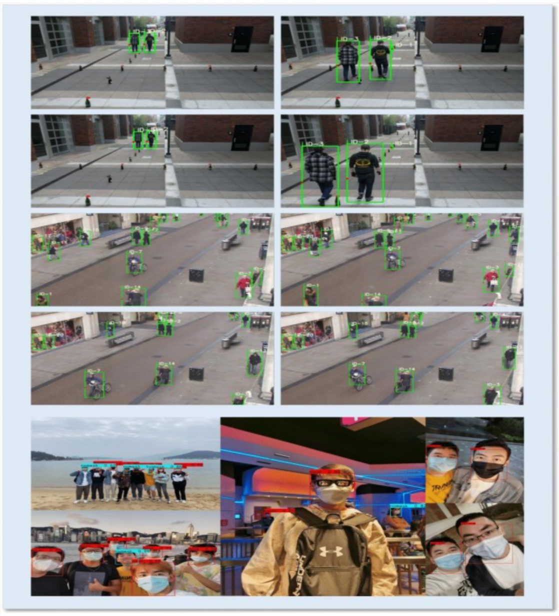


Figure 9. The experimental result of our improved model.

4.2. *System performance*

As shown in Fig.10, we recorded the operation of each part of the system when the IoT system was running. When the system starts to run the IoT system and activates a node, the GPU resources consumed are not large, and as the mask detection begins, the GPU and CPU speed up. Also always filled. However, compared with the previous v5 model, the detection effect can make better use of gpu resources. In the same case of full load, our method is faster and has a higher frame rate. Under the same limited computing power, our improved model can support detection of more objects.

And we need less computing power for the same detection effect, which allows our model to independently detect other objects concurrently or run two improved models at the same time.

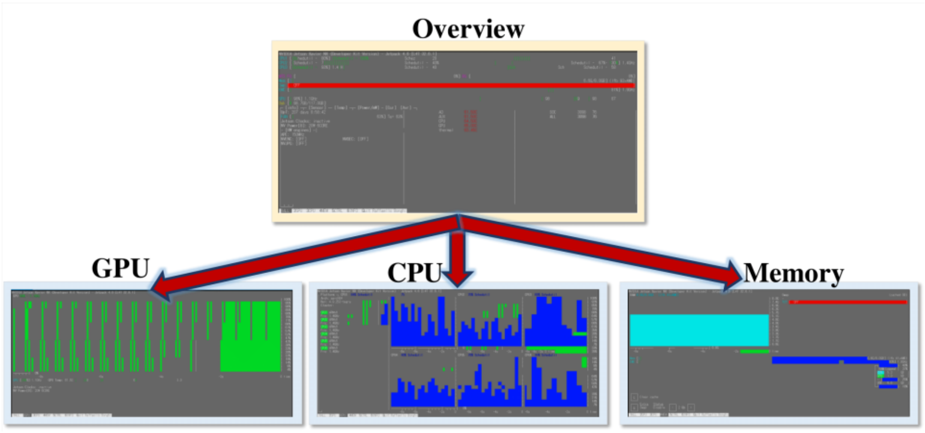


Figure 10. The system performance of our method.

5. Conclusion

In this paper, we successfully combine an IoT system and an improved YOLOv5 model for mask detection in daily life. Our method can well assist government personnel and antiepidemic workers to count the wearing of masks, so as to formulate effective anti-epidemic policies. We use the lightweight model YOLOv5 and combine Ghost-Net and BiFPN on it, which enables it to exert more computing power with less computing resources. The improved YOLOv5 combined with our deployed IoT system can better perform concurrent operations, and different edge nodes can run different projects at the same time, such as vehicle counting, person tracking, mask detection, etc. Furthermore, this IoT system combined with YOLOv5 is easy to expand and deploy, and it is very easy to clone on the server. Its lightweight setting can help people maximize efficiency under the condition of limited computing power.

References

[1] L. Chettri and R. Bera, ”A Comprehensive Survey on Internet of Things (IoT) Toward 5G Wireless Systems,” in IEEE Internet of Things Journal, vol. 7, no. 1, pp. 16-32, Jan. 2020.

[2] G. Xu et al., ”TT-SVD: An Efficient Sparse Decision-Making Model With Two-Way Trust Recommendation in the AI-Enabled IoT Systems,” in IEEE Internet of Things Journal, vol. 8, no. 12, pp. 9559-9567, 15 June15, 2021.

[3] J. Hwang, L. Nkenyereye, N. Sung, J. Kim and J. Song, ”IoT Service Slicing and Task Offloading for Edge Computing,” in IEEE Internet of Things Journal, vol. 8, no. 14, pp. 11526-11547, 15 July15, 2021.

[4] F. Hussain, R. Hussain, S. A. Hassan and E. Hossain, ”Machine Learning in IoT Security: Current Solutions and Future Challenges,” in IEEE Communications Surveys & Tutorials, vol. 22, no. 3, pp. 1686-1721, thirdquarter 2020.

[5] B. Yuan, J. Wang, P. Wu and X. Qing, ”IoT Malware Classification Based on Lightweight Convolutional Neural Networks,” in IEEE Internet of Things Journal, vol. 9, no. 5, pp. 3770-3783, 1 March1, 2022.

[6] B. Mao, Y. Kawamoto and N. Kato, ”AI-Based Joint Optimization of QoS and Security for 6G Energy Harvesting Internet of Things,” in IEEE Internet of Things Journal, vol. 7, no. 8, pp. 7032-7042, Aug. 2020.

[7] R. He, J. Cao, L. Song, Z. Sun and T. Tan, ”Adversarial Cross-Spectral Face Completion for NIR-VIS Face Recognition,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 5, pp. 1025- 1037, 1 May 2020.

[8] C. Fu, X. Wu, Y. Hu, H. Huang and R. He, ”DVG-Face: Dual Variational Generation for Heterogeneous Face Recognition,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 6, pp. 2938- 2952, 1 June 2022.

[9] K. He, G. Gkioxari, P. Dollar and R. Girshick, ”Mask R-CNN,” in IEEE ´ Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 2, pp. 386-397, 1 Feb. 2020.

[10] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh and J. Liang, ”UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation,” in IEEE Transactions on Medical Imaging, vol. 39, no. 6, pp. 1856-1867, June 2020.

[11] C. Fan, J. Yi, J. Tao, Z. Tian, B. Liu and Z. Wen, ”Gated Recurrent Fusion With Joint Training Framework for Robust End-to-End Speech Recognition,” in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 29, pp. 198-209, 2021.

[12] L. Chai, J. Du, Q. -F. Liu and C. -H. Lee, ”A Cross-Entropy-Guided Measure (CEGM) for Assessing Speech Recognition Performance and Optimizing DNN-Based Speech Enhancement,” in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 29, pp. 106-117, 2021.

[13] Z. Zhang, B. Zhong, S. Zhang, Z. Tang, X. Liu and Z. Zhang, ”Distractor-Aware Fast Tracking via Dynamic Convolutions and MOT Philosophy,” 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 1024-1033.

[14] N. Wang, W. Zhou, J. Wang and H. Li, ”Transformer Meets Tracker: Exploiting Temporal Context for Robust Visual Tracking,” 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 1571-1580.

[15] YOLOv5, https://github.com/ultralytics/yolov5.
[16] M. Tan, R. Pang and Q. V. Le, ”EfficientDet: Scalable and Efficient Object Detection,” 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 10778-10787.
[17] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu and C. Xu, ”GhostNet: More Features From Cheap Operations,” 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 1577- 1586.
[18] Kubemetes, https://kubernetes.io/.
[19] L. -C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, ”DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 834-848, 1 April 2018.
[20] L. Sifre and S. Mallat, ”Rigid-motion scattering for texture classification”, arXiv:1403.1687 [cs], Mar. 2014.
[21] A. Howard et al., ”Searching for MobileNetV3,” 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 1314- 1324.
[22] S. Liu, L. Qi, H. Qin, J. Shi and J. Jia, ”Path Aggregation Network for Instance Segmentation,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 8759-8768.
[23] T. -Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan and S. Belongie, ´ ”Feature Pyramid Networks for Object Detection,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 936-944, doi: 10.1109/CVPR.2017.106.
[24] Real-World Masked Face Dataset, https://github.com/Xzhangyang/Real-World-Masked-Face-Dataset.