# YOLO-Head: An Input Adaptive Neural Network Preprocessor

Biao Hou, Shenxuan Zhou, Xiaoyu Chen, Heng Jiang, Hao Wang

# YOLO-Head:An Input Adaptive Neural Network Preprocessor

Biao Hou, Shenxuan Zhou, Xiaoyu Chen, Heng Jiang, and Hao Wang

Xidian University, Xi'an 710071, China
sxzhou@stu.xidian.edu.cn

**Abstract.** Over the past decade, object detectors have demonstrated remarkable performance in various applications, such as traffic monitoring, customer tracking, and surveillance. Although advanced lightweight models have been proved to have ultra real-time speed on GPU, in edge-based video analytics system, edge servers are usually embedded devices with NPU which support general neural network operators. When we implemented deep learning models on embedded devices, images usually need to be preprocessed to the network input size. This leads to the common target detectors not being end-to-end. Image preprocessing is not the key problem of real-time inferencing on devices with high-performance CPU, but the same algorithm will bring noticeable delay on embedded devices. To overcome this, we designed YOLO-Head, a module that can handle the input of arbitrarily size according to general neural network operators. Experiment results show that YOLO-Head achieves significant (60.89%) speed improvement when 1080p image zooms to 640 x 640. Furthermore, YOLOv5-S detector with adaptive head can effectively solve the delay problem due to the image resize. The frame rate improves to 35.2 FPS, approximately 6 times faster than the convectional method in video stream processing on RK3588.

**Keywords:** Video analytics, Deep neural networks, Object detection

## 1 Introduction

Video cameras are pervasive in today's society, with cities and organizations steadily increasing the size and reach of their deployments.[1] Key to video stream processing applications has been recent advances in deep learning which has obtained high accuracy in multiple scenes for object detection and recognition. In a typical real-time video analytics pipeline[2], a camera streams live video to cloud servers, which immediately run object detection models (e.g., YOLO[3]) to answer user queries about that video. Cloud-based video analytics requires a lot of computing resources and network resources. The end-to-end frame rate needs to be more than 30fps for real-time video streams in that case the network delay of information transmission can not be ignored.

Therefore, a large number of embedded devices on the edge side are added in edge-based video analytics paradigm[7]. They are deployed near mobile devices, with small network delay. These devices are equipped with NPU instead

of GPU, which execute neural operators quite efficiently as well. On the neural network arithmetic unit, most general neural network computing modules have been implemented, but special types of algorithms are difficult to implement such as image resize. On the other side, it is usually necessary for preprocessor to adjust the data size from image input to neural network[4]. It is mush slower than neural network computing modules.

Image scaling is usually accomplished by interpolations. In digital signal processing, it can be defined as the convolution of the continuous impulse response of a discrete image signal with a two-dimensional reconstructed filter. Continuous images can be reconstructed with appropriate window functions, e.g., rectangular windows(nearest neighbor interpolation), triangular functions(linear interpolation)[9]. The algorithm based on region mainly uses mean filter, which replaces the original pixel value with the average of all the pixels in the template(area interpolation). Although different scale algorithms employ different strategies, their speeds on ARM-based processors can not meet the real-time requirements as shown in figure 2.

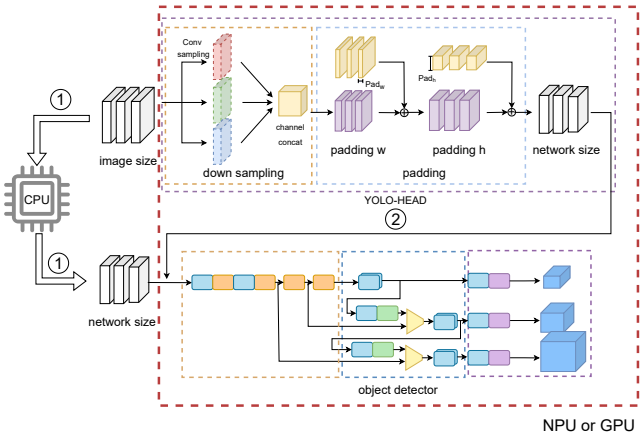


**Fig. 1.** In the popular target detection framework, it is usually necessary to use the interpolation algorithm in the CPU to scale the input image to the network input, as shown in way 1. In embedded devices, the relevant algorithms for scaling are quite time-consuming. We propose an adaptive head to complete data preprocessing making data flow always on NPU or GPU as shown in way 2.

One of the existing methods is to do two-dimensional image operation by designing special integrated circuits, while it is complicated and difficult to implement for ordinary developers[5, 8]. From the perspective of software, we propose an algorithm instead of resize, which can be widely used in embedded development boards with NPU. We propose the adaptive head module solving the delay of its preprocessing, which is the key point to promote the efficiency of detector. Our contributions can be summarized as follows:

- **Convolution sampling unit** We use convolution as adaptive down sampling unit, which has very excellent performance on NPU. With the complement unit composed of concat operation, arbitrarily scale can be completed. When the 1080p image is scaled to 640x640, the image preprocessing time is decreased to 60.89% by YOLO-Head.
- **Target Detector General Component** YOLO-Head can be added to most target detectors to form an input adaptive model. Experiments on YOLOv5-S show that our proposed method improves video stream processing speed by approximately six times.

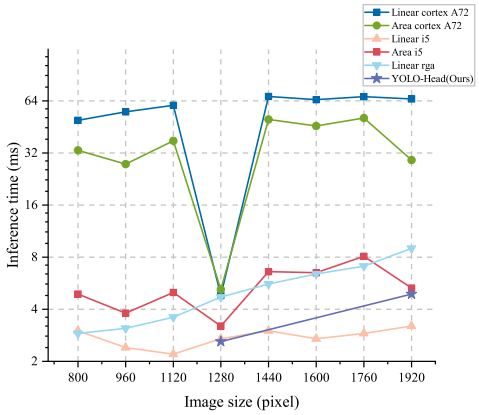


**Fig. 2.** Zoom the picture from image size to 640x640 pixels. The proposed YOLO-Head outperforms a range of state-of-the-art algorithms.

## 2 Method

### 2.1 YOLO-Head

Generally speaking, in the real-time video analytics pipelines, the image size in the video stream is fixed depends on cameras. With the development of photoelectric imaging, image size is generally more than 1080x1920 pixels. Due to the computing resources of devices, the input of deep learning network can not be as large as the video size. Therefore, it is necessary to resize the image to the network before inference. Figure 2 illustrates that the time required for this resize operation on personal computer (i5-6300HQ CPU) is less than 8ms. On high-performance processors resize algorithm is less affected by the origin size, but it will bring serious delay on embedded device. In particular, in some divisible sizes, different resize algorithms will be equivalent to certain algorithm.

According to the documentation of opencv[6], it is most precise to shrink the image using area interpolation, while in the case of magnifying cubic interpolation works best. Cubicis replaced by a slightly inferior linear algorithm because

of its slow speed. In this paper, if there is no additional explanation, the resizing algorithm based on area interpolation is discussed.

The overall structure of YOLO-Head is shown in Figure 1. It is equivalent to area interpolation and is designed to replace resize operation. Common operators such as conv(convolution) or pooling(averagepooling) can be used when down sampling data. Figure 3 is a convolution sampling operator to double the down sampling. Average pooling works the same as convolution sampling operator. Conv is selected as the down sampling unit in YOLO-Head, which is more efficient on RK3588. The lower sampling coefficient is:

$$scale = \min\left(\frac{net_w}{img_w}, \frac{net_h}{img_h}\right) \tag{1}$$

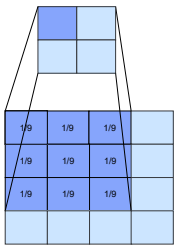


**Fig. 3.** Conv operator accomplish down sampling.


**Fig. 4.** Resize by aligning the upper left corner when $img_w$ is not a multiple of $net_w$(or $img_h$ is not a multiple of $net_h$).

In order to further reduce the computation of the model, the top-left edge alignment is adopted to handle the case where $1scale$ is not an integer, which has little impact on post-processing and is shown in Figure 4. A compensation module is added to YOLO-Head to solve the problem of misalignment. The compensation values for height and width is

$$\begin{cases} pad_h = net_h - \frac{img_h}{scale} \\ pad_w = net_w - \frac{img_w}{scale} \end{cases} \tag{2}$$

After adding the adaptive head in front of the detector, the procedure of resizing the image to the network size will be omitted. Arbitrary images can be

directly transmitted to the network. In addition, the implementation of resize algorithm on embedded devices may have slight differences, which may reduce the accuracy of model. The improved detector maintains the consistency of the algorithm on the inferencing side and training side.

### 2.2 Pipeline

**Base detector.** The classic YOLOv5-S is selected as our default object detector. It inherits and carries forward YOLO series with several tricks, e.g., adaptive anchor, strong data argumentations, advanced network structure and outstanding loss function. Considering the handware implementation of NPU operator, we have modified the large step maxpooling operator, slice operator and activation function in Focus module. Max pooling layers with small steps in series instead of large steps are used because Max pooling with large steps on NPU takes a long time. The slice operator is also poorly implemented on some RKNN devices. We replaced it by a conv with a special weight distribution. The above two optimization strategies speed up without changing the runtime results. Since the convolution, Relu and BN layers can be combined on NPU devices, but special activation functions such as Silu are not supported, all activation functions are replaced with Relu. In addition, the data post-processing part of the model has serious quantification accuracy problems and has to be removed from the YOLO model.

**Adaptive head.** During the translational deployment of the target detector, the input adaptive network designed by the input parameters of the task will added in front of the YOLO detector. Deployment details of YOLO-Head is depicted in Algorithm 1. Experiments show that the concat operator used in the adaptive head often runs on the CPU, which greatly increases the time consumed. Therefore, the specific deployment scheme is to use a simplified YOLO-Head which has no concat operations after image compensated to an integer multiple of the network input. Our strategy will not affect the results of YOLO-Head.

**Train & Inference.** The dataset, which is taken with the same camera, has the same picture size. According to equation 1 and 2, our adaptive head is constructed and added to the YOLO detector. Since the weight data of adaptive network is constant, it will not affect the training results. In inferencing, our improved YOLOv5-S model can complete end-to-end feature extraction only by inputting source image. The real-time target prediction frames are obtained when data decoding and NMS are performed after feature results are transmitted to CPU.

**Algorithm 1** Adaptive head deployment process

**Input:** A bitmap Im of size $Img_w \times Img_h$
**Output:** A bitmap $Im_{resize}$ of size $Net_w \times Net_h$
*// Calculate Construction parameters scale and ($pad_h$, $pad_w$)*
Update scale based on Equation (1)
Update $pad_h$ and $pad_w$ based on Equation (2)
*// scaling in the original ratio*
**for** $c$ in output_channels **do**
    sample_unit = Conv2d(input_c=1,output_c=1,kernal=scale)
    $Im_{resize}[c]$ = sample_unit($Im[c]$)
*// two-dimensional padding*
$tmp_h = \frac{Img_h}{scale}$, $tmp_w = \frac{Img_w}{scale}$
**if** $pad_w != 0$ **then**
    $padding = zeros_{1\times3\times tmp_h \times pad_w}$
    concat($Im_{resize}$, $padding$)
**if** $pad_h != 0$ **then**
    $padding = zeros_{1\times3\times Net_w \times pad_w}$
    concat($Im_{resize}$, $padding$)

## 3 Experiments

### 3.1 Settings

**Datasets.** The experiment was carried out on our own infrared autopilot dataset including diverse urban outdoor scenes in Xi'an city. It has high frame rate(30 FPS) and high resolution(1080x1920 pixels) sensor data. The dataset is divided into two parts: training set and verification set. The verification set has 5 videos, with a total of 42378 frames.

**Implementation details.** If not specified, we use YOLOv5-S as our default detector. The network input is 640x640 pixels while the picture size in the video stream is 1080x1920 pixels. We trained on 4x3090ti GPUs and got weight file. Because only RKNN model can be loaded on RK3588, we convert the weight file into RKNN model through onnx transition. In inferencing evaluation, we run on a single NPU of RK3588, including data post-processing and NMS time.

### 3.2 Evaluation for YOLO-Head

According to the deployment strategy of YOLO-Head in Algorithm 1, only the image size of an integer multiple of the network input size needs to be calculated during evaluation. The image of 1280x1280 pixels uses our input adaptive module resize to 640x640, which takes an average of 2.6ms, while the image of 1920x1920 pixels takes 4.9ms. 1920x1920 images use area interpolation on cortex A72 needs 29.1ms. The run time improvement comes from making full use of the parallel computing power of NPU and reduce the operation of CPU. Our method shows strong competitiveness on the embedded platform and solves the problem that the image preprocessing is time-consuming in the mobile terminal.

### 3.3 Application in object detector

The YOLOv5 source model is struggling when inferencing on RK3588. Table 1 shows after adopting the NPU optimization strategy, the modified YOLOv5 model has a slight loss in accuracy (reduced by 4.4%), but the running time of a single frame is reduced by 60.89%. Although the single frame operation result is acceptable, in real-time stream processing, image preprocessing and detection result post-processing bring serious delay.

Table 2 shows after using our adaptive input module, the FPS is increased from the original 5.78 frame/s to 37.59, which effectively reduces the delay caused by preprocessing. Our YOLO-Head can achieve the performance similar to that of RGA(a CPU-independent graphics acceleration engine RGA on RK3399/3588, with basic operations of 2D images). When there is no RGA on the general embedded device, our method can be used to replace it equivalently on NPU. When the error loss is allowed, YOLO-Head can be easily added before the trained model, which eliminates the step of retraining the deep learning model.

**Table 1.** YOLOv5-s single frame inference results on RK3588(single core).

| Model | Small stride | Slice | Relu | YOLO-Head | Inference time(ms) | AP |
|---|---|---|---|---|---|---|
| YOLOv5-S | | | | | 165.31 | 87.6% |
| | ✓ | ✓ | | | 64.65 | 87.1% |
| | ✓ | ✓ | ✓ | | 45.18 | 82.4% |
| | ✓ | ✓ | ✓ | ✓ | 49.07 | 82.3% |

**Table 2.** Real time video stream processing on RK3588(single core).

| SYSTEM | YOLO-Head | RGA (linear) | FPS | AP |
|---|---|---|---|---|
| YOLOv5-S (NPU modify) | | | 5.78 | 82.4% |
| | | ✓ | 37.59 | 81.9% |
| | ✓ | | 35.20 | 82.3% |

## 4 Conclusion

This paper presents an adaptive head module called YOLO-Head to handle the time-consuming problem of scaling on embedded devices. Our adaptive head is composed of general neural network operators, which is simple, fast and accurate. The idea can be widely applied to the deep learning model. Our initial experiments are encouraging and effectively solves the time delay problem of image preprocessing on embedded devices.

## 5 Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 62171347, 61877066, 61771379, 62001355, 62101405; the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant 61621005; the Key Research and Development Program in Shaanxi Province of China under Grant 2019ZDLGY03-05 and 2021ZDLGY02-08; the Science and Technology Program in Xi'an of China under Grant XA2020-RGZNTJ-0021; 111 Project.

## References

1. Li, Y., Padmanabhan, A., Zhao, P., Wang, Y., Xu, G., Ravi, N: Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In:Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, pp. 359-376 (2020)
2. Can 30,000 Cameras Help Solve Chicago's Crime Problem?,https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html.
3. Redmon, J., Farhadi, A: YOLO9000: Better, Faster, Stronger. In CVPR (2017)
4. Redmon, J., Farhadi, A: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
5. Open source warehouse for acceleration engine RGA,https://github.com/rockchip-linux/linux-rga. Accessed 26 June 2021
6. OpenCV: Geometric Image Transformations, https://docs.opencv.org/4.x/d9/df8/tutorial_root.html. Accessed 10 May 2022
7. Chen, J., Ran X.: Deep learning with edge computing: A review. Proceedings of the IEEE. 107(8): 1655-1674 (2019)
8. Wang, P., Cao Y., Ding, M., Zhang Z., Qu J.: A SoC collaborative accelerated design method of image scaling algorithm. Journal of Beijing University of Aeronautics and Astronautics. 45(02):333-339 (2019)
9. Huang, Y.: Research on Image Scaling Algorithms. Master,HeFei University of Technology (2010)