2. Related work

2.1. Real-time Object Detectors

The current mainstream real-time object detectors are the YOLO series [2, 7, 13–15, 25, 30, 31, 47–49, 61–63, 74, 75], and most of these models use CSPNet [64] or ELAN [65] and their variants as the main computing units. In terms of feature integration, improved PAN [37] or FPN [35] is often used as a tool, and then improved YOLOv3 head [49] or FCOS head [57, 58] is used as prediction head. Recently some real-time object detectors, such as RT DETR [43], which puts its fundation on DETR [4], have also been proposed. However, since it is extremely difficult for DETR series object detector to be applied to new domains without a corresponding domain pre-trained model, the most widely used real-time object detector at present is still YOLO series. This paper chooses YOLOv7 [63], which has been proven effective in a variety of computer vision tasks and various scenarios, as a base to develop the proposed method. We use GELAN to improve the architecture and the training process with the proposed PGI. The above novel approach makes the proposed YOLOv9 the top real-time object detector of the new generation.

2.2. Reversible Architectures

The operation unit of reversible architectures [3, 16, 19] must maintain the characteristics of reversible conversion, so it can be ensured that the output feature map of each layer of operation unit can retain complete original information. Before, RevCol [3] generalizes traditional reversible unit to multiple levels, and in doing so can expand the semantic levels expressed by different layer units. Through a literature review of various neural network architectures, we found that there are many high-performing architectures with varying degree of reversible properties. For example, Res2Net module [11] combines different input partitions with the next partition in a hierarchical manner, and concatenates all converted partitions before passing them backwards. CBNet [34, 39] re-introduces the original input data through composite backbone to obtain complete original information, and obtains different levels of multi-level reversible information through various composition methods. These network architectures generally have excellent parameter utilization, but the extra composite layers cause slow inference speeds. DynamicDet [36] combines CBNet [34] and the high-efficiency real-time object detector YOLOv7 [63] to achieve a very good trade-off among speed, number of parameters, and accuracy. This paper introduces the DynamicDet architecture as the basis for designing reversible branches. In addition, reversible information is further introduced into the proposed PGI. The proposed new architecture does not require additional connections during the inference process, so it can fully retain the advantages of speed, parameter amount, and accuracy.

2.3. Auxiliary Supervision

Deep supervision [28, 54, 68] is the most common auxiliary supervision method, which performs training by inserting additional prediction layers in the middle layers. Especially the application of multi-layer decoders introduced in the transformer-based methods is the most common one. Another common auxiliary supervision method is to utilize the relevant meta information to guide the feature maps produced by the intermediate layers and make them have the properties required by the target tasks [18, 20, 24, 29, 76]. Examples of this type include using segmentation loss or depth loss to enhance the accuracy of object detectors. Recently, there are many reports in the literature [53, 67, 82] that use different label assignment methods to generate different auxiliary supervision mechanisms to speed up the convergence speed of the model and improve the robustness at the same time. However, the auxiliary supervision mechanism is usually only applicable to large models, so when it is applied to lightweight models, it is easy to cause an under parameterization phenomenon, which makes the performance worse. The PGI we proposed designed a way to reprogram multi-level semantic information, and this design allows lightweight models to also benefit from the auxiliary supervision mechanism.

3. Problem Statement

Usually, people attribute the difficulty of deep neural network convergence problem due to factors such as gradient vanish or gradient saturation, and these phenomena do exist in traditional deep neural networks. However, modern deep neural networks have already fundamentally solved the above problem by designing various normalization and activation functions. Nevertheless, deep neural networks still have the problem of slow convergence or poor convergence results.

In this paper, we explore the nature of the above issue further. Through in-depth analysis of information bottleneck, we deduced that the root cause of this problem is that the initial gradient originally coming from a very deep network has lost a lot of information needed to achieve the goal soon after it is transmitted. In order to confirm this inference, we feedforward deep networks of different architectures with initial weights, and then visualize and illustrate them in Figure 2. Obviously, PlainNet has lost a lot of important information required for object detection in deep layers. As for the proportion of important information that ResNet, CSPNet, and GELAN can retain, it is indeed positively related to the accuracy that can be obtained after training. We further design reversible network-based methods to solve the causes of the above problems. In this section we shall elaborate our analysis of information bottleneck principle and reversible functions.

3.1. Information Bottleneck Principle

According to information bottleneck principle, we know that data X may cause information loss when going through transformation, as shown in Eq. 1 below:

$$I(X, X) \geq I(X, f_\theta(X)) \geq I(X, g_\phi(f_\theta(X))), \quad (1)$$

where I indicates mutual information, f and g are transformation functions, and θ and ϕ are parameters of f and g, respectively.

In deep neural networks, $f_\theta(\cdot)$ and $g_\phi(\cdot)$ respectively represent the operations of two consecutive layers in deep neural network. From Eq. 1, we can predict that as the number of network layer becomes deeper, the original data will be more likely to be lost. However, the parameters of the deep neural network are based on the output of the network as well as the given target, and then update the network after generating new gradients by calculating the loss function. As one can imagine, the output of a deeper neural network is less able to retain complete information about the prediction target. This will make it possible to use incomplete information during network training, resulting in unreliable gradients and poor convergence.

One way to solve the above problem is to directly increase the size of the model. When we use a large number of parameters to construct a model, it is more capable of performing a more complete transformation of the data. The above approach allows even if information is lost during the data feedforward process, there is still a chance to retain enough information to perform the mapping to the target. The above phenomenon explains why the width is more important than the depth in most modern models. However, the above conclusion cannot fundamentally solve the problem of unreliable gradients in very deep neural network. Below, we will introduce how to use reversible functions to solve problems and conduct relative analysis.

3.2. Reversible Functions

When a function r has an inverse transformation function v, we call this function reversible function, as shown in Eq. 2.

$$X = v_\zeta(r_\psi(X)), \quad (2)$$

where ψ and ζ are parameters of r and v, respectively. Data X is converted by reversible function without losing information, as shown in Eq. 3.

$$I(X, X) = I(X, r_\psi(X)) = I(X, v_\zeta(r_\psi(X))). \quad (3)$$

When the network's transformation function is composed of reversible functions, more reliable gradients can be obtained to update the model. Almost all of today's popular deep learning methods are architectures that conform to the reversible property, such as Eq. 4.

$$X^{l+1} = X^l + f_\theta^{l+1}(X^l), \quad (4)$$

where l indicates the l-th layer of a PreAct ResNet and f is the transformation function of the l-th layer. PreAct ResNet [22] repeatedly passes the original data X to subsequent layers in an explicit way. Although such a design can make a deep neural network with more than a thousand layers converge very well, it destroys an important reason why we need deep neural networks. That is, for difficult problems, it is difficult for us to directly find simple mapping functions to map data to targets. This also explains why PreAct ResNet performs worse than ResNet [21] when the number of layers is small.

In addition, we tried to use masked modeling that allowed the transformer model to achieve significant breakthroughs. We use approximation methods, such as Eq. 5, to try to find the inverse transformation v of r, so that the transformed features can retain enough information using sparse features. The form of Eq. 5 is as follows:

$$X = v_\zeta(r_\psi(X) \cdot M), \quad (5)$$

where M is a dynamic binary mask. Other methods that are commonly used to perform the above tasks are diffusion model and variational autoencoder, and they both have the function of finding the inverse function. However, when we apply the above approach to a lightweight model, there will be defects because the lightweight model will be under parameterized to a large amount of raw data. Because of the above reason, important information $I(Y, X)$ that maps data X to target Y will also face the same problem. For this issue, we will explore it using the concept of information bottleneck [59]. The formula for information bottleneck is as follows:

$$I(X, X) \geq I(Y, X) \geq I(Y, f_\theta(X)) \geq ... \geq I(Y, \hat{Y}). \quad (6)$$

Generally speaking, $I(Y, X)$ will only occupy a very small part of $I(X, X)$. However, it is critical to the target mission. Therefore, even if the amount of information lost in the feedforward stage is not significant, as long as $I(Y, X)$ is covered, the training effect will be greatly affected. The lightweight model itself is in an under parameterized state, so it is easy to lose a lot of important information in the feedforward stage. Therefore, our goal for the lightweight model is how to accurately filter $I(Y, X)$ from $I(X, X)$. As for fully preserving the information of X, that is difficult to achieve. Based on the above analysis, we hope to propose a new deep neural network training method that can not only generate reliable gradients to update the model, but also be suitable for shallow and lightweight neural networks.

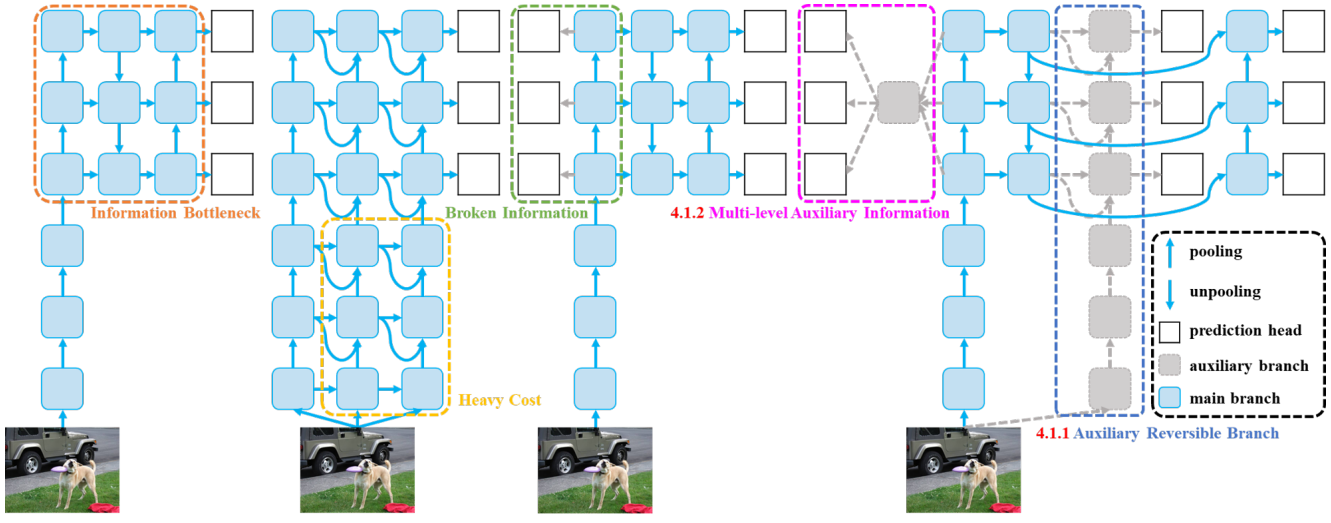

Figure 3. PGI and related network architectures and methods. (a) Path Aggregation Network (PAN)) [37], (b) Reversible Columns (RevCol) [3], (c) conventional deep supervision, and (d) our proposed Programmable Gradient Information (PGI). PGI is mainly composed of three components: (1) main branch: architecture used for inference, (2) auxiliary reversible branch: generate reliable gradients to supply main branch for backward transmission, and (3) multi-level auxiliary information: control main branch learning plannable multi-level of semantic information.

4. Methodology

4.1. Programmable Gradient Information

In order to solve the aforementioned problems, we propose a new auxiliary supervision framework called Programmable Gradient Information (PGI), as shown in Figure 3 (d). PGI mainly includes three components, namely (1) main branch, (2) auxiliary reversible branch, and (3) multi-level auxiliary information. From Figure 3 (d) we see that the inference process of PGI only uses main branch and therefore does not require any additional inference cost. As for the other two components, they are used to solve or slow down several important issues in deep learning methods. Among them, auxiliary reversible branch is designed to deal with the problems caused by the deepening of neural networks. Network deepening will cause information bottleneck, which will make the loss function unable to generate reliable gradients. As for multi-level auxiliary information, it is designed to handle the error accumulation problem caused by deep supervision, especially for the architecture and lightweight model of multiple prediction branch. Next, we will introduce these two components step by step.

4.1.1 Auxiliary Reversible Branch

In PGI, we propose auxiliary reversible branch to generate reliable gradients and update network parameters. By providing information that maps from data to targets, the loss function can provide guidance and avoid the possibility of finding false correlations from incomplete feedforward features that are less relevant to the target. We propose the maintenance of complete information by introducing reversible architecture, but adding main branch to reversible architecture will consume a lot of inference costs. We analyzed the architecture of Figure 3 (b) and found that when additional connections from deep to shallow layers are added, the inference time will increase by 20%. When we repeatedly add the input data to the high-resolution computing layer of the network (yellow box), the inference time even exceeds twice the time.

Since our goal is to use reversible architecture to obtain reliable gradients, "reversible" is not the only necessary condition in the inference stage. In view of this, we regard reversible branch as an expansion of deep supervision branch, and then design auxiliary reversible branch, as shown in Figure 3 (d). As for the main branch deep features that would have lost important information due to information bottleneck, they will be able to receive reliable gradient information from the auxiliary reversible branch. These gradient information will drive parameter learning to assist in extracting correct and important information, and the above actions can enable the main branch to obtain features that are more effective for the target task. Moreover, the reversible architecture performs worse on shallow networks than on general networks because complex tasks require conversion in deeper networks. Our proposed method does not force the main branch to retain complete original information but updates it by generating useful gradient through the auxiliary supervision mechanism. The advantage of this design is that the proposed method can also be applied to shallower networks.