electronics

MDPI

Article

Accelerating Die Bond Quality Detection Using Lightweight Architecture DSGβSI-Yolov7-Tiny

Bao Rong Chang [1], Hsiu-Fen Tsai [2,*] and Wei-Shun Chang [1]

[1] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 81148, Taiwan; brchang@nuk.edu.tw (B.R.C.); m1125513@mail.nuk.edu.tw (W.-S.C.)
[2] Department of Fragrance and Cosmetic Science, Kaohsiung Medical University, Kaohsiung 80708, Taiwan
* Correspondence: sftsai@kmu.edu.tw

Abstract: The die bonding process is one of the most critical steps in the front-end semiconductor packaging process, as it significantly affects the yield of the entire IC packaging process. This research aims to find an efficient, intelligent vision detection model to identify whether each chip correctly adheres to the IC substrate; by utilizing the detection model to classify the type of defects occurring in the die bond images, the engineers can analyze the leading causes, enabling timely adjustments to key machine parameters in real-time, improving the yield of the die bond process, and significantly reducing manufacturing cost losses. This study proposes the lightweight Yolov7-tiny model using Depthwise-Separable and Ghost Convolutions and Sigmoid Linear Unit with β parameter (DSGβSI-Yolov7-tiny), which we can apply for real-time and efficient detection and prediction of die bond quality. The model achieves a maximum FPS of 192.3, a precision of 99.1%, and an F1-score of 0.97. Therefore, the performance of the proposed DSGβSI-Yolov7-tiny model outperforms other methods.

Keywords: Yolov7; ghost convolution; depthwise-separable convolution; object detection; image recognition; ModifiedSiLU; AdaptiveSiLU

Citation: Chang, B.R.; Tsai, H.-F.; Chang, W.-S. Accelerating Die Bond Quality Detection Using Lightweight Architecture DSGβSI-Yolov7-Tiny. *Electronics* **2024**, *13*, 4573. https://doi.org/10.3390/electronics13224573

Academic Editors: Hyeonjoon Moon and Lien Minh Dang

Received: 22 October 2024
Revised: 11 November 2024
Accepted: 17 November 2024
Published: 20 November 2024

1. Introduction

Figure 1 illustrates the complete IC packaging and testing process. AFE stands for Assembly Front End, ABE represents Assembly Back End, FT refers to Final Test, WT is Wafer Test, and PA indicates Pre-assembly. Die bond is one of the steps in the IC packaging and testing process. In this process, Fab/BL (foundry) initially provides the wafer, and then the IC packaging and testing factory tests the wafer to know any defects. Next, in the pre-assembly phase, the wafer is cut to prepare for the subsequent assembly steps. Moving into AFE, the die bond step involves securely attaching the die (bare die) to the IC substrate. The main focus of this paper is to acquire the images of die bonds from the machine and use an intelligent vision detection model to detect whether each chip correctly adheres to the IC substrate. In the wire bond phase, the die is connected to the IC substrate via bonding wires, allowing the electrical signals of a die to transmit to external circuits. Afterward, the process moves into ABE, where molding encapsulates the die with epoxy resin to protect it. Marking involves imprinting identification marks on the package exterior, and plating applies surface treatments such as gold or tin to enhance electrical connectivity and trim/form cuts. It shapes the packaged components into their final form. Finally, the product undergoes testing and packaging. The entire process, including wafer fabrication, testing, packaging, and shipping, constitutes the typical flow of semiconductor production, as shown in Figure 1.

Die bonding is the process of cutting a wafer into individual dies and attaching each die to the IC substrate using conductive mediums such as glue or gold balls, with glue being the most commonly used material. This process includes several steps: preparing the substrate, applying adhesive to the package base, positioning the die, applying heat

and pressure, cooling and curing, and testing and packaging. Die bonding technology is widely used in electronic packaging, module manufacturing, and optical component manufacturing. It is one of the most critical processes in front-end semiconductor packaging, as it directly affects the quality of the entire IC packaging process. Real-time detection and prediction of die bond yield are essential for adjusting machine production settings, significantly improving the die bond yield [1] and reducing manufacturing costs. This study used images from a renowned semiconductor company in southern Taiwan for data preprocessing to create the training and testing datasets. People manually inspected the collected data and evaluated each of the four sides and four corners of any single image for adhesion quality. Fab then categorized these images into bond_good (including side_good and corner_good), representing properly adhered dies, and bond_bad (including side_bad and corner_bad), representing incorrectly adhered dies. To further determine which side or corner was not fully adhered to, a hexadecimal type system was used, with two bits per group, to specify die bond type.

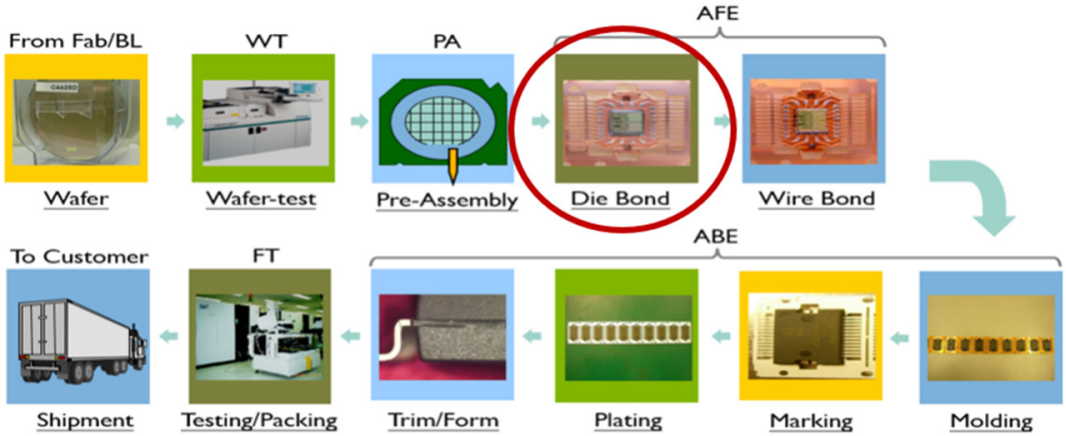


Figure 1. IC packaging and testing process.

Therefore, this paper aims to accelerate the calculation of the visual detection and prediction of die bonds to achieve lightweight models and to try new activation functions to improve the accuracy of detection and prediction to ensure the effectiveness of the die bond process. This study refers to the related algorithm improved by Yolov7-tiny [2]. We also explore the improved method of Yolov4-tiny in convolution calculation [3] and the lightweight architecture of Yolov5 for fast wafer contour detection [4]. This study further understands the application case of Yolov7-VD for the intelligent visual detection of vehicles [5] and the practical process of die bond automatic optical inspection (AOI) and identification methods [6]. Therefore, to detect and predict whether the die adherence is complete, this study has proposed Yolov4-tiny, Yolov5n [7], Yolov7, Yolov7-tiny, DSG-Yolov7, DSG-Yolov7-tiny, DSGSI-Yolov7-tiny, and DSGβSI-Yolov7-tiny models for a comparison of execution performance. Finally, the model DSGβSI-Yolov7-tiny, with the best execution performance, was selected, and the best-trained model was exported to TensorFlow Lite and then integrated into the control system of the die bond machine. The most significant contribution of this study is the use of the optimal detection system to determine which type of bond_bad it is. After summarizing the reasons, the engineers can promptly adjust the critical parameters of the machine online to reduce the occurrence of electrical faults or increased resistance. This approach can improve the die bond process yield and significantly reduce manufacturing cost losses.

2. Related Work

2.1. Literature Review

Applications of convolutional neural networks (CNNs) for object detection and image recognition, developed on powerful computing platforms, often face challenges when

deployed on embedded platforms for portable devices. These platforms typically have limited hardware resources, preventing the applications from performing as expected to complete tasks. A key challenge is lightening models while maintaining a certain level of precision to accelerate execution speed, making them practical and efficient when deployed on embedded platforms.

Sandler et al. [8] discussed MobileNet, a lightweight deep neural network (DNN) model with fewer parameters and reduced computational complexity. Additionally, Howard et al. [9] studied how MobileNets utilize depthwise separable convolutions (DSCs), which consist of 1×1 pointwise convolutions (PWCs) and depthwise convolutions (DWCs), to reduce model complexity. This decomposition significantly reduces the amount of computation and the number of parameters, improving computational efficiency several times compared to standard convolutions. Furthermore, Hsu et al. [10] explored how depthwise separable convolutions enhance computational efficiency and model performance in dense prediction tasks. Depthwise separable convolutions can improve the computational speed of the model and achieve better prediction results while maintaining a small model size.

Additionally, Zhang et al. [11] combined Yolov5 and GhostNet to detect and identify seven types of orchard pests in real-time, using feature maps, heatmaps, and loss curves to explain the advantages of their method. Smaller neural networks are better suited for deployment on FPGA and other memory-limited embedded devices. This research provides a method for deploying algorithms on embedded devices. Sun et al. [12] explored the integration of the Ghost module into the Yolov7 architecture, creating a lightweight and efficient object detection model called Ghost-YoLov7-SIoU. Ghost-YoLov7-SIoU integrates Ghost modules into the backbone and neck to replace some traditional layers. Experimental results demonstrated the effectiveness of this method in improving detection efficiency while ensuring detection precision.

Lang et al. [13] studied the Yolov7-tiny architecture, which achieves a lightweight design by streamlining the original Yolov7 backbone, neck, and head, making it more suitable for running on resource-constrained devices. The integration of GhostNet further improved the computational efficiency of the model, significantly reducing the computational and storage requirements while maintaining high performance. Addressing the issues of inaccurate human and vehicle detection and slow detection speeds in nighttime scenarios, where Yolov7-tiny fails to meet the demands, Yang et al. [14] proposed adding the Ghostnet V2 module to the Yolov7-tiny backbone to reduce parameters. They also replaced the LeakyReLU activation function in the convolutional layers with the FReLU function. The model introduced the omni-dimensional dynamic convolution (ODConv) module, the C3 module, and the parameter-free attention module SimAM. Experimental results showed an inference speed of 47 fps, a 0.64% decrease in mAP, and a 59% reduction in floating-point operations, demonstrating significant performance improvements. Regarding the other detection methods, Shafiee Sarvestani et al. [15] proposed incorporating the generated crack maps into classic quality assessment (QA) models, enabling many advances in applying 3D textured meshes. Song et al. [16] explored end-to-end pedestrian detection and focused on training a pedestrian detection model by discarding non-maximum suppression (NMS) post-processing.

2.2. The Yolov4-Tiny Model

The Yolov4-tiny [3] model is a streamlined version of Yolov4, designed to be a more lightweight model specifically for devices requiring fast inference and limited resources, such as mobile devices and embedded systems. Compared to Yolov4, the Yolov4-tiny model simplifies its structure by reducing convolutional layers and feature pyramids to lower computational costs. Its backbone is CSPDarknet53-tiny, incorporating the CSPNet (Cross-Stage Partial Network) design into the Darknet network to enhance inference speed while maintaining a certain level of precision. This design allows the model to retain the high-efficiency feature extraction capabilities of Yolov4 while reducing the number of parameters.

Yolov4-tiny, with its lower complexity, can achieve real-time object detection. Its prominent architecture includes two feature extraction layers. After extracting features through CSPDarknet53-tiny, they are input into two different-sized convolutional kernels, producing two sets of feature maps. These feature maps, through anchor boxes at different scales, are used to predict bounding boxes for detecting objects of varying sizes. Although Yolov4-tiny sacrifices some precision compared to the full Yolov4 model, it offers significant advantages in real-time performance and efficiency, making it well-suited for fast detection scenarios such as vehicle tracking, surveillance systems, or other resource-constrained applications, as shown in Figure 2. In Figure 2, the input is a single source image, including three red, green, and blue images, and the output is the object detection and classification. This paper has defined the input/output of yolo-related models in the same way in all the following sections.

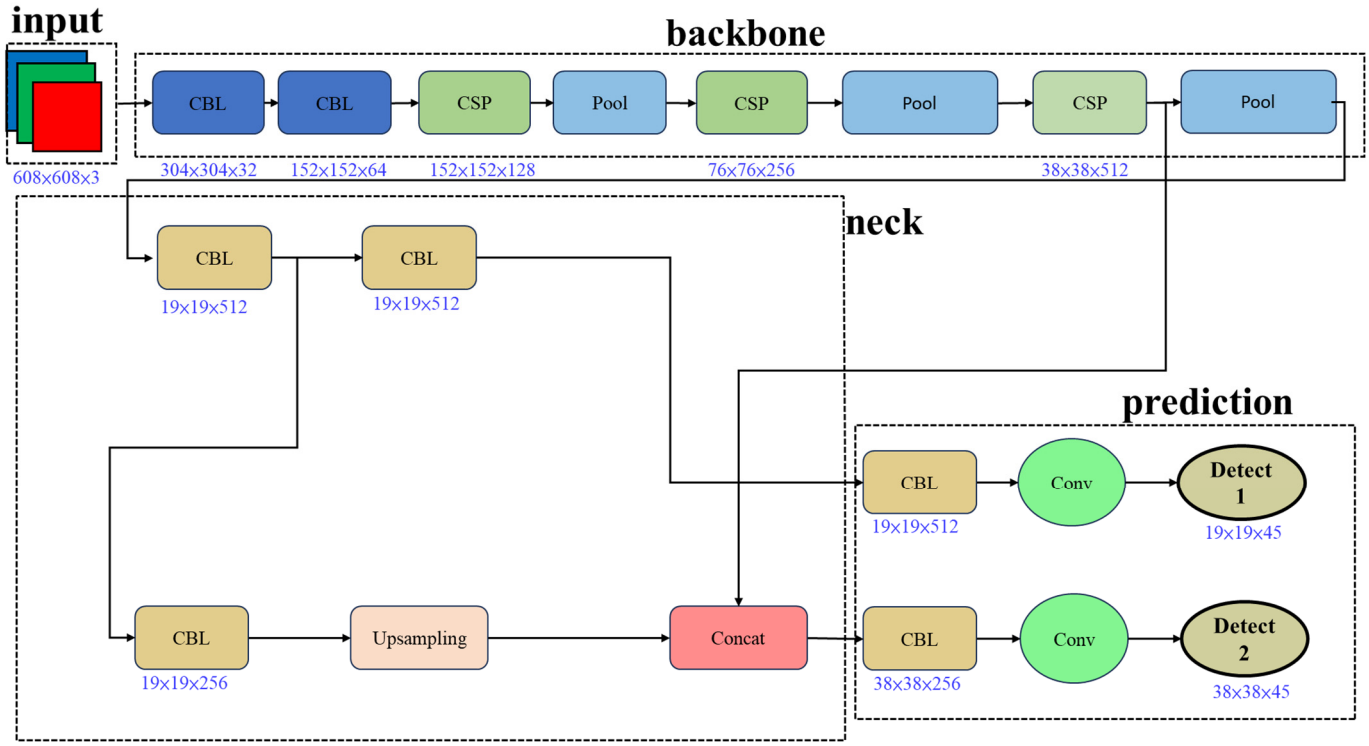


Figure 2. Yolov4-tiny architecture.

2.3. The Yolov5n Model

Yolov5n [7] is the "nano" version of the Yolov5 series, designed to be even more lightweight than Yolov5m and Yolov5s. Its goal is to optimize inference speed further and reduce the model size to suit extremely resource-limited application scenarios. It is the smallest, fastest, and most parameter-efficient model in the Yolov5 family. It is ideal for deployment on edge devices, Internet of Things (IoT) devices, and low-power embedded systems.

Yolov5n inherits the core architecture of Yolov5, incorporating CSPNet as its backbone to enhance feature extraction efficiency. It also utilizes a Focus layer to transform spatial information from input images into more useful low-level features. Compared to other Yolov5 versions, Yolov5n significantly reduces the number of parameters and model size, enabling ultra-fast inference while maintaining essential detection precision. The model employs multi-scale feature fusion techniques, such as Feature Pyramid Networks (FPN) and Path Aggregation Networks (PANet), to enhance detection capabilities for objects of various sizes.

Yolov5n also supports automatic anchor adjustment, mixed-precision training, and data augmentation techniques, such as mosaic data augmentation, to improve the model's

training effectiveness and inference efficiency. While Yolov5n is the lightest model in the Yolov5 series, its precision performance decreases slightly compared to other Yolov5 models. However, due to its lightweight architecture, it offers the fastest speed among them. Its lightweight characteristics allow it to operate in limited hardware environments and achieve real-time object detection, which is crucial for highly resource-constrained scenarios, as shown in Figure 3.

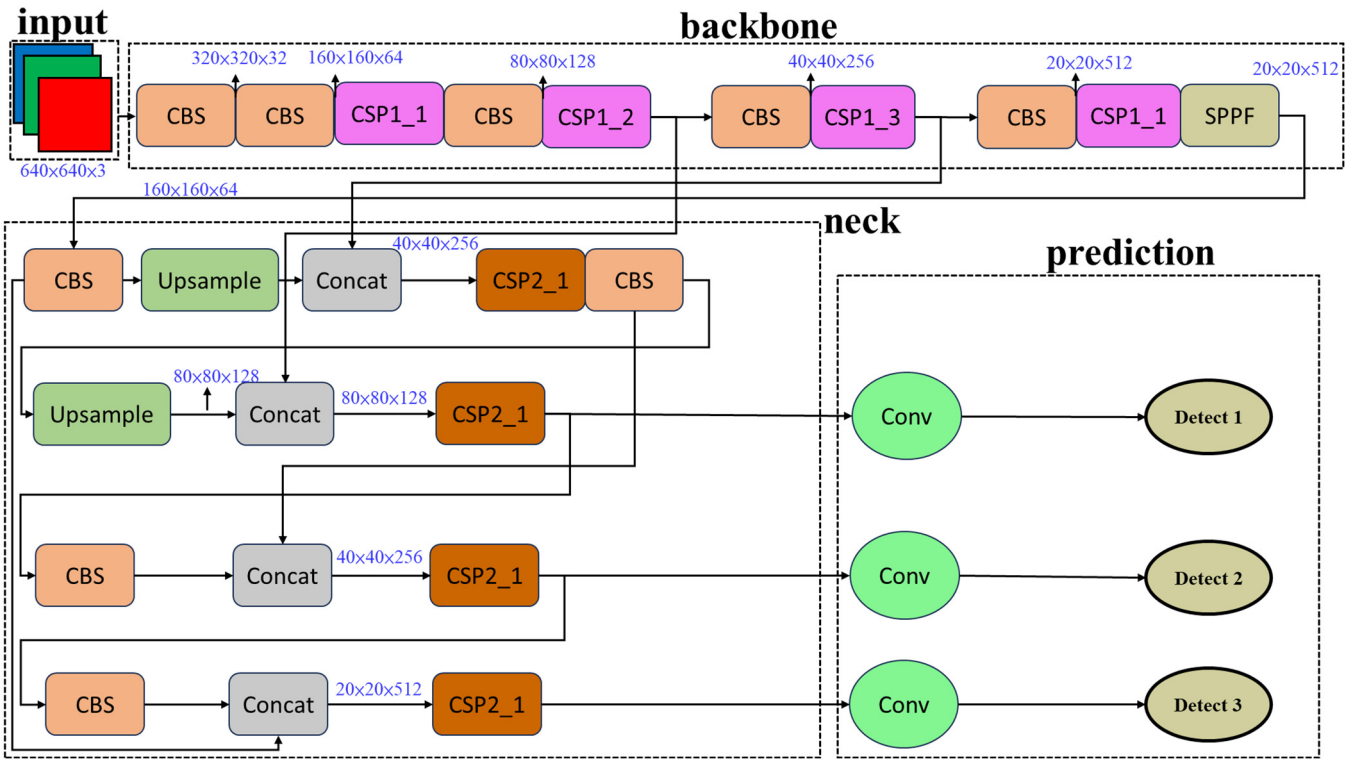


Figure 3. Yolov5n architecture.

2.4. The Yolov7 Model

Yolov7 [5] is one of the most powerful versions in the Yolo series, featuring significant optimizations in speed and precision compared to previous Yolo versions. Yolov7 employs multiple innovative technologies, including the Extended Efficient Layer Aggregation Networks (E-ELAN) architecture, designed to improve feature fusion efficiency and model expressiveness. This design enhances the depth and width of the network structure, allowing for better extraction of multi-scale features and improved detection performance for complex scenes and small targets.

Additionally, Yolov7 introduces a model reparameterization technique (RepConv), which enables the model to use complex convolutional structures during optimization training while converting to simplified structures during inference. This fact reduces computational overhead and increases inference speed, greatly enhancing efficiency while maintaining high precision. Yolov7 also incorporates a label assignment strategy, which adaptively assigns positive and negative sample labels, improving detection performance for targets of varying scales. It also supports dynamic label assignment (Dynamic Head), enhancing the flexibility of the detection heads.

Yolov7 demonstrates superior performance on multiple public datasets (e.g., the COCO dataset), achieving high inference speeds while maintaining excellent precision, making it suitable for deployment on resource-constrained edge devices. Compared to previous Yolov4 and Yolov5 versions, Yolov7 has significantly improved efficiency, making it ideal for real-time object detection applications. In summary, Yolov7's advanced architecture design, reparameterization techniques, and efficient label assignment strategies

enable it to achieve dual enhancements in speed and precision, making it one of the most efficient object detection models currently available, as shown in Figure 4.

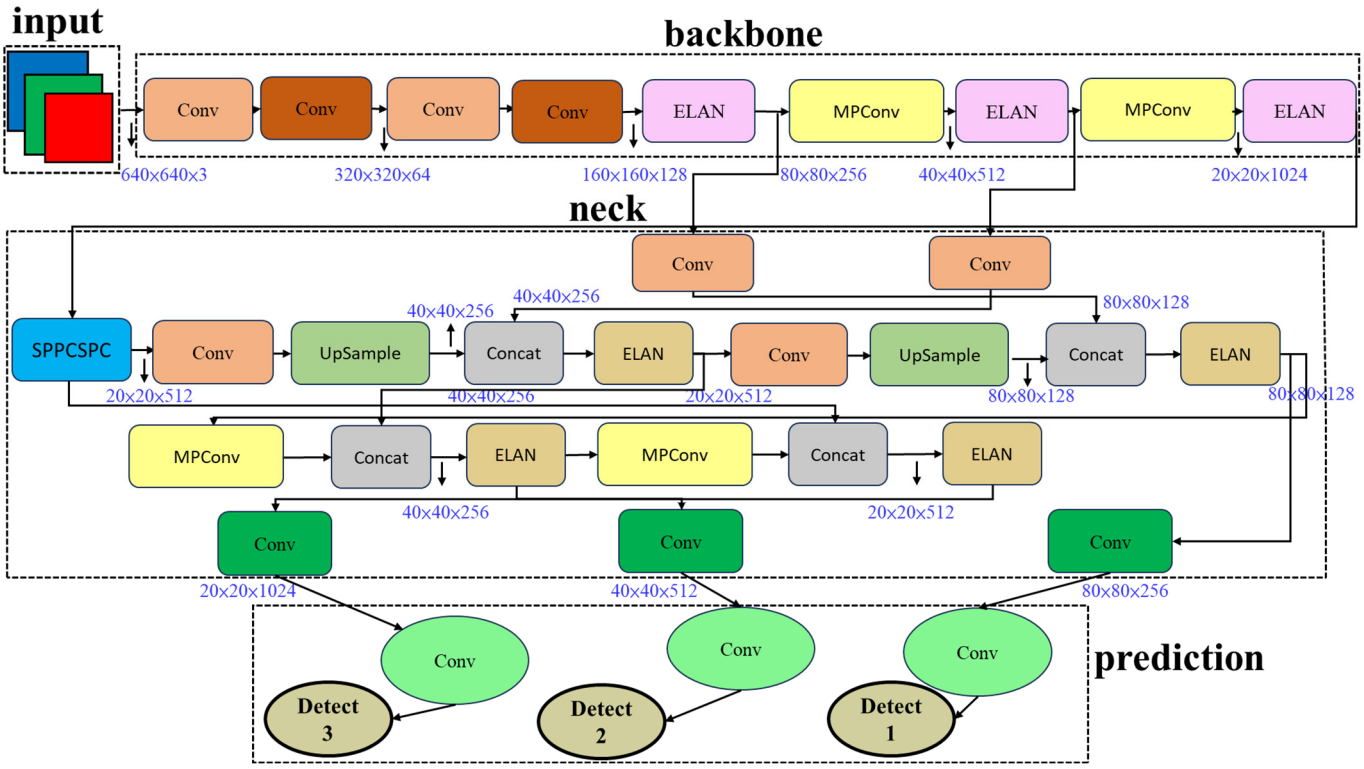


Figure 4. Yolov7 architecture.

2.5. The Yolov7-Tiny Model

Yolov7-tiny [2] is a lightweight object detection model in the Yolo series, designed specifically for mobile devices or resource-constrained environments. Compared to the full-size version of Yolov7, Yolov7-tiny incorporates several design simplifications to reduce computational resource requirements and enhance running speed. Yolov7-tiny features fewer layers and parameters, typically including simplified convolutional and pooling layers, making it lighter than the standard Yolov7 model. This simplification alters the depth and detail of feature extraction, thereby reducing the computational burden.

Moreover, Yolov7-tiny employs fewer convolutional layers and introduces lighter convolution operations in certain areas, such as depthwise separable or Ghost convolutions, to further decrease computational demands. These modifications enable the model to achieve faster inference speeds while maintaining a relatively low level of precision. Overall, Yolov7-tiny significantly reduces computational complexity and model size while retaining a certain level of detection capability, making it perform better in real-time applications, as shown in Figure 5.

2.6. The DSG-Yolov7 Model

Figure 6 illustrates that the core of Ghost convolution (G Conv) [17] lies in introducing the Ghost module, which reduces computational load by generating many virtual feature maps. These virtual feature maps are created through linear combinations of a few actual convolution results, significantly decreasing computational and storage demands while retaining the main features. Additionally, Ghost convolution dramatically reduces the number of parameters that need to be trained compared to traditional convolution layers. This modification makes the model more lightweight and suitable for resource-constrained environments. Ghost convolution can provide a higher feature representation capability

at the exact computational cost. The model can perform well when dealing with complex scenarios by generating more feature maps without excessive computational overhead.

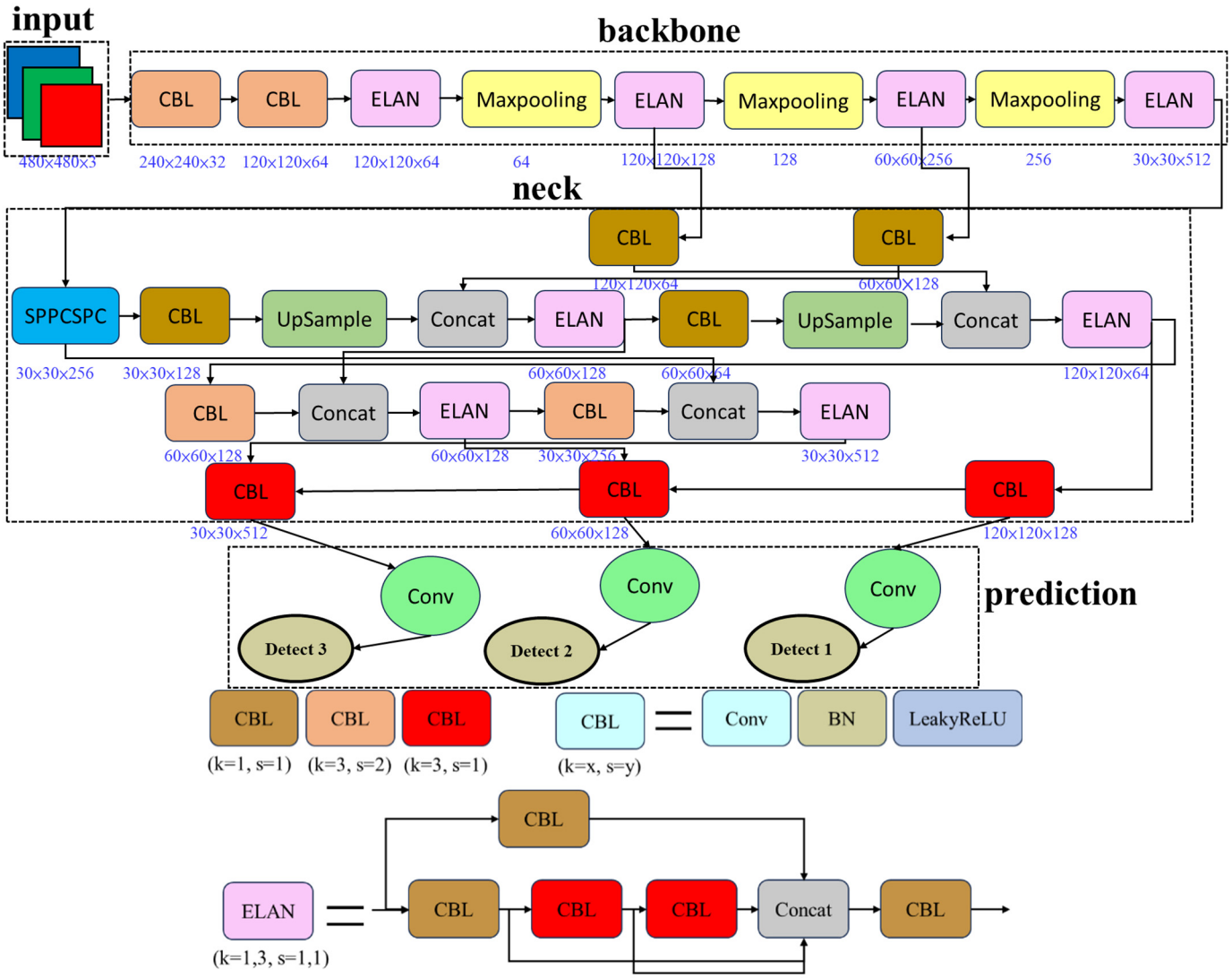


Figure 5. Yolov7-tiny architecture. Note: k represents kernel size, and s stands for stride.

Figure 7 shows that depthwise separable convolution (DS Conv) [18] is a simple and efficient convolution operation primarily used to reduce convolutional neural networks' computational complexity and parameter count. The main benefits of applying depthwise separable convolution include significantly reducing computation and parameter count. Unlike standard convolution, which performs a single convolution operation, depthwise separable convolution breaks it down into depthwise convolution and then pointwise convolution, thus significantly reducing computational complexity. For example, in standard convolution, computational complexity increases linearly with the number of input channels, output channels, and kernel size, whereas, in depthwise separable convolution, this manner reduces complexity to the combination of depthwise and pointwise convolutions, which is especially advantageous for resource-limited environments such as mobile devices. Furthermore, this method reduces the number of parameters in the network, helping to lower the risk of model overfitting and improving inference speed.

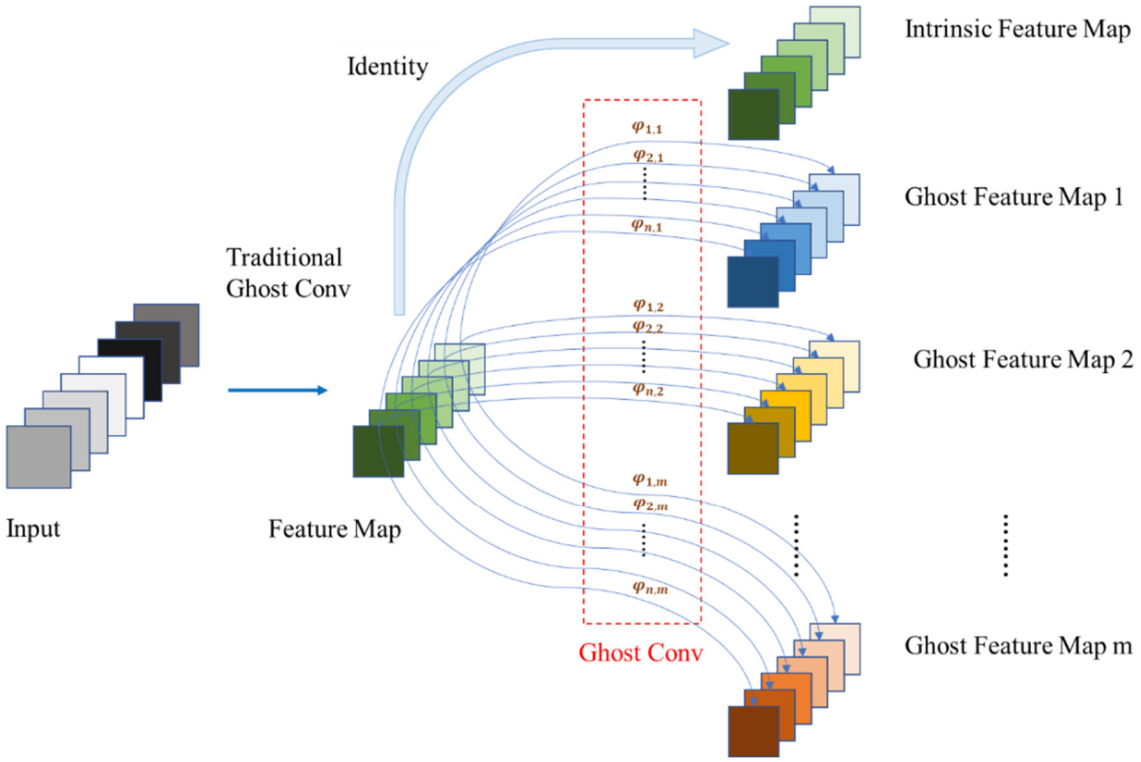


Figure 6. Ghost convolution.

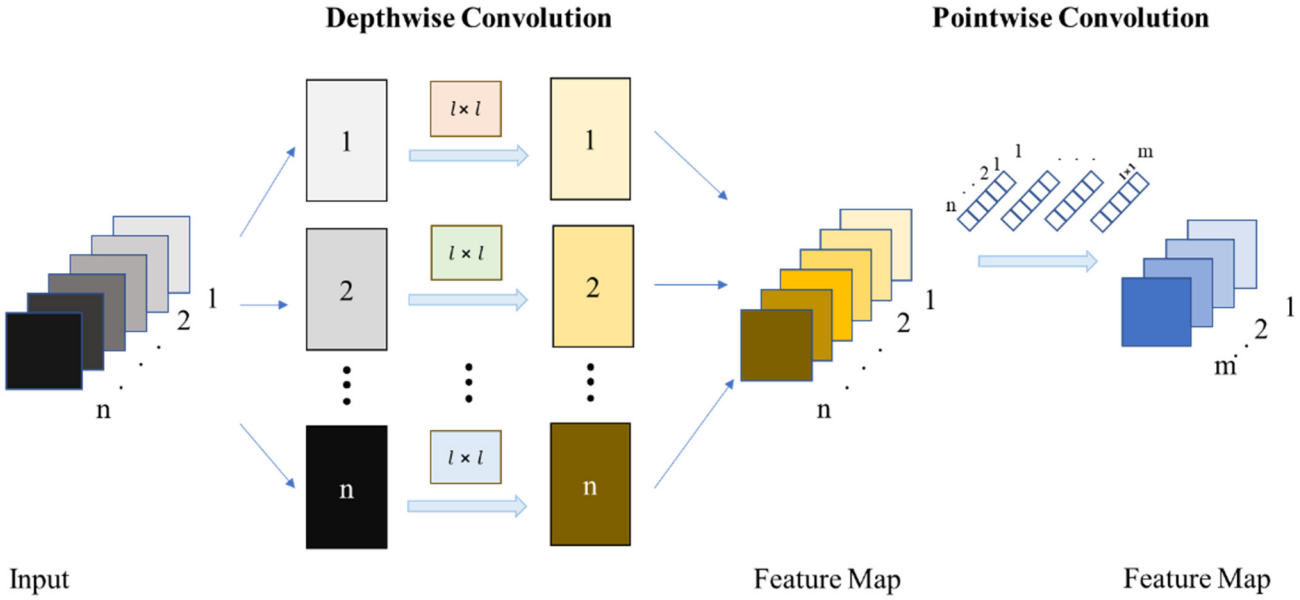


Figure 7. Depthwise separable convolution.

Simplifying convolution calculations enables the model to be lightweight; therefore, the convolution operation that combines depthwise separable convolution with Ghost convolution [3] is abbreviated as DSG Conv to implement simplified convolution calculations, as shown in Figure 8. Inspired by model design [10,13], replacing some convolution layers in Yolov7 with DSGConv layers can achieve a lightweight model and effectively enhance the inference speed, as illustrated in Figure 9.

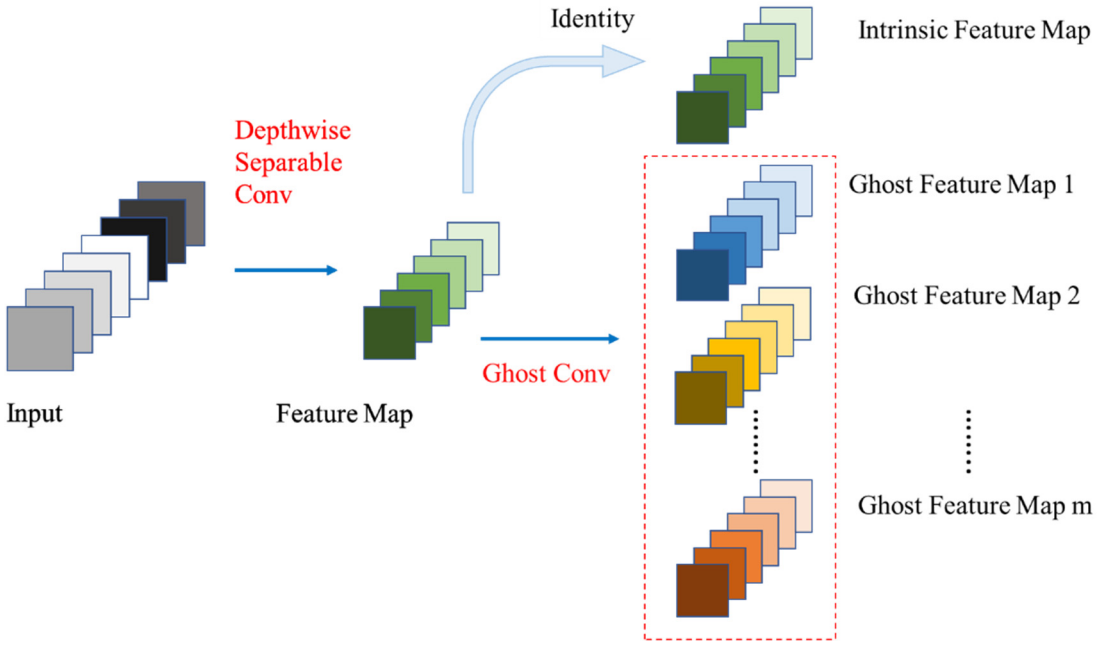


Figure 8. Depthwise separable and Ghost convolutions.

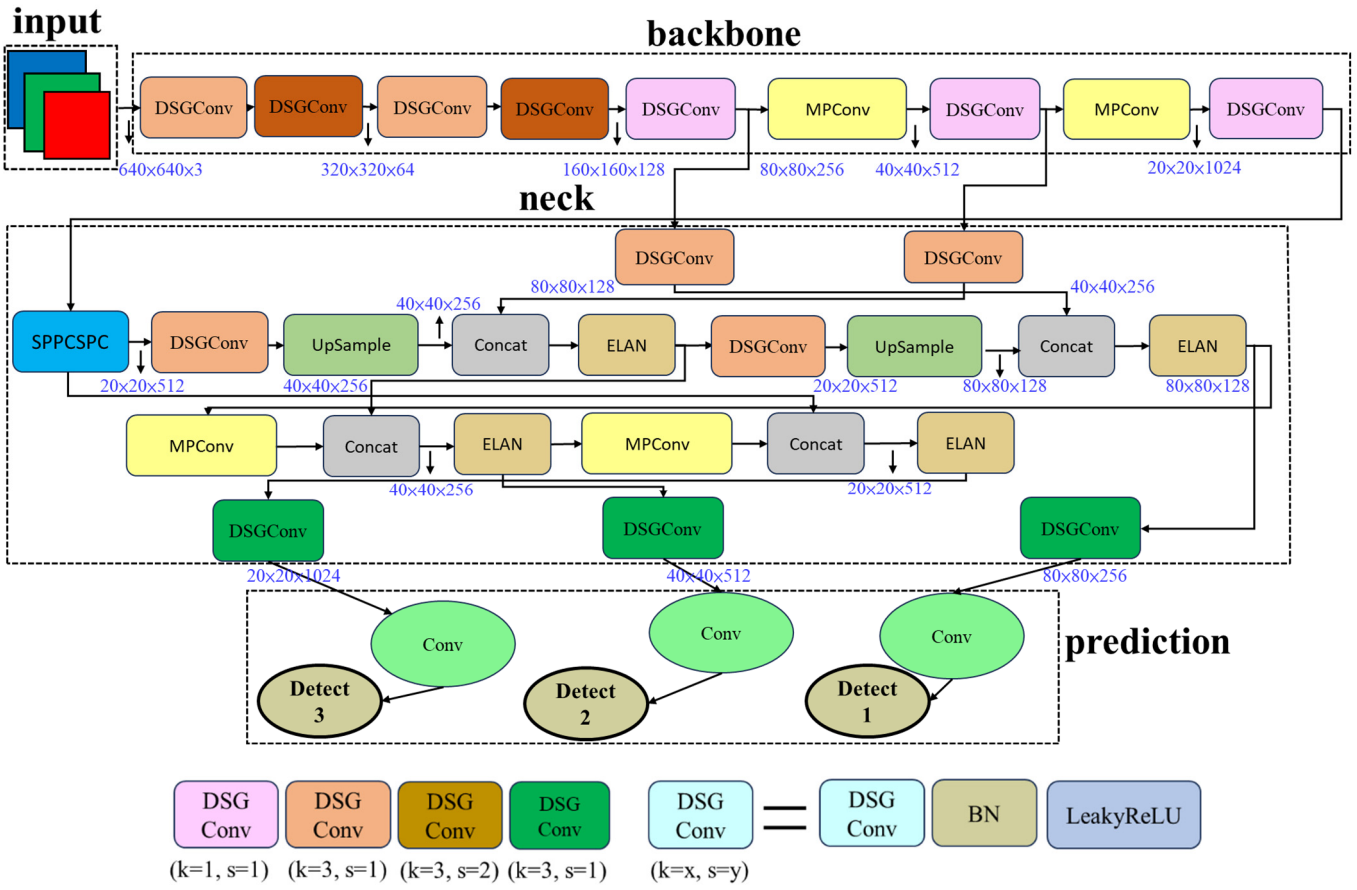


Figure 9. DSG-Yolov7 architecture. Note: k represents kernel size, and s stands for stride.

3. Methods

This chapter's methodology involves preparing the data, conducting preprocessing, selecting the model, setting parameters, and then training. Next, we can examine the results of these trained models, such as the confusion matrix, PR curve, and loss graph,

to determine whether they met the training objectives. After that, we will test the die bond image recognition by classifying the die bond adhesion quality into bond_good and bond_bad categories. Finally, we will evaluate and compare the performance of each model based on their performance metric.

3.1. Data Collection and Preprocessing

The image data from a well-known semiconductor manufacturer in southern Taiwan were collected from a top-down view of the machines, as shown in Figure 10. After annotation, we can generate corresponding XML label files and categorize them into side_good, side_bad, corner_good, and corner_bad. Then, the procedure converts the XML label files into the VOC label format to serve as the input data format for the YOLO-related models. For cross-dataset evaluation, a total of 3145 images were collected, divided into training data (2204 images), validation data (467 images) from machine #1, and testing data (474 images) from machine #2, with a distribution ratio of approximately 70%, 15%, and 15%, respectively.

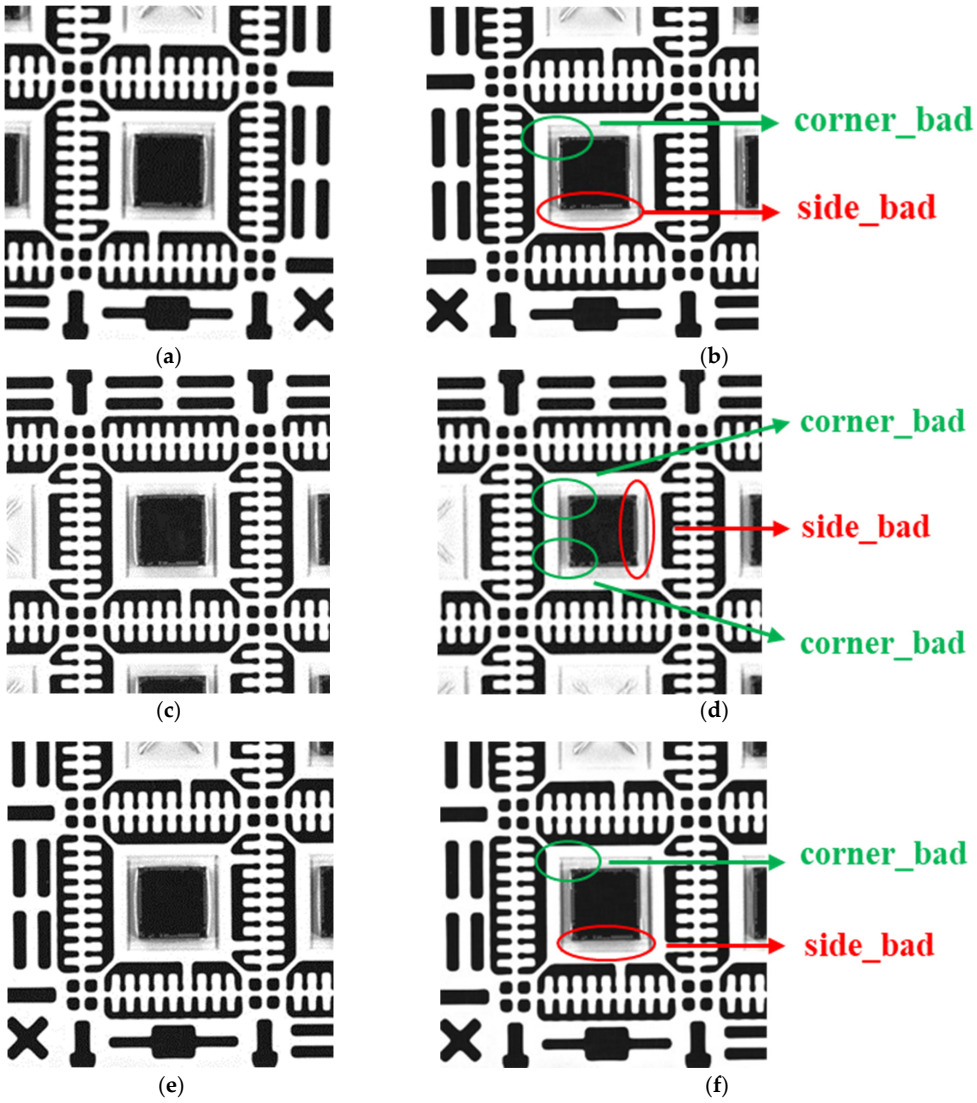


Figure 10. Sample images of die bond categories. (**a**,**c**,**e**) bond_good; (**b**,**d**,**f**) bond_bad.

This study first trained the models for Yolov4-tiny, Yolov5n, Yolov7, and Yolov7-tiny versions. Before training, the procedure must set hyperparameters. This study set the parameter epoch to 120, batch size to 16, and input image size to 1024 × 1024. The epoch indicates the number of complete iterations over the dataset for training. The batch size

refers to the number of samples used to update the model weights after each training iteration. The specification of input image size accelerates object detection and image recognition speed for the die bond. Similarly, this study applies the training process to the DSG-Yolov7, DSG-Yolov7-tiny, and DSGSI-Yolov7-tiny lightweight models.

3.2. Recognition Data

During the testing phase, this study input the test set of 474 images into the trained models for recognition, yielding identification results for each image in the test set. Each image's recognition results include a category and a confidence level. The categories include side_good, side_bad, corner_good, and corner_bad. The confidence level indicates the model's certainty in its prediction. Experimental results show that the confidence levels for each part of all images are at least 95% or higher. If the die bond meets the conditions for complete adhesion on all four die sides, the recognition result will indicate side_good; otherwise, it will indicate side_bad. Similarly, if the die bond meets the conditions for complete adhesion at all four die corners, it will display corner_good; otherwise, it will show corner_bad, as illustrated in Figure 11.

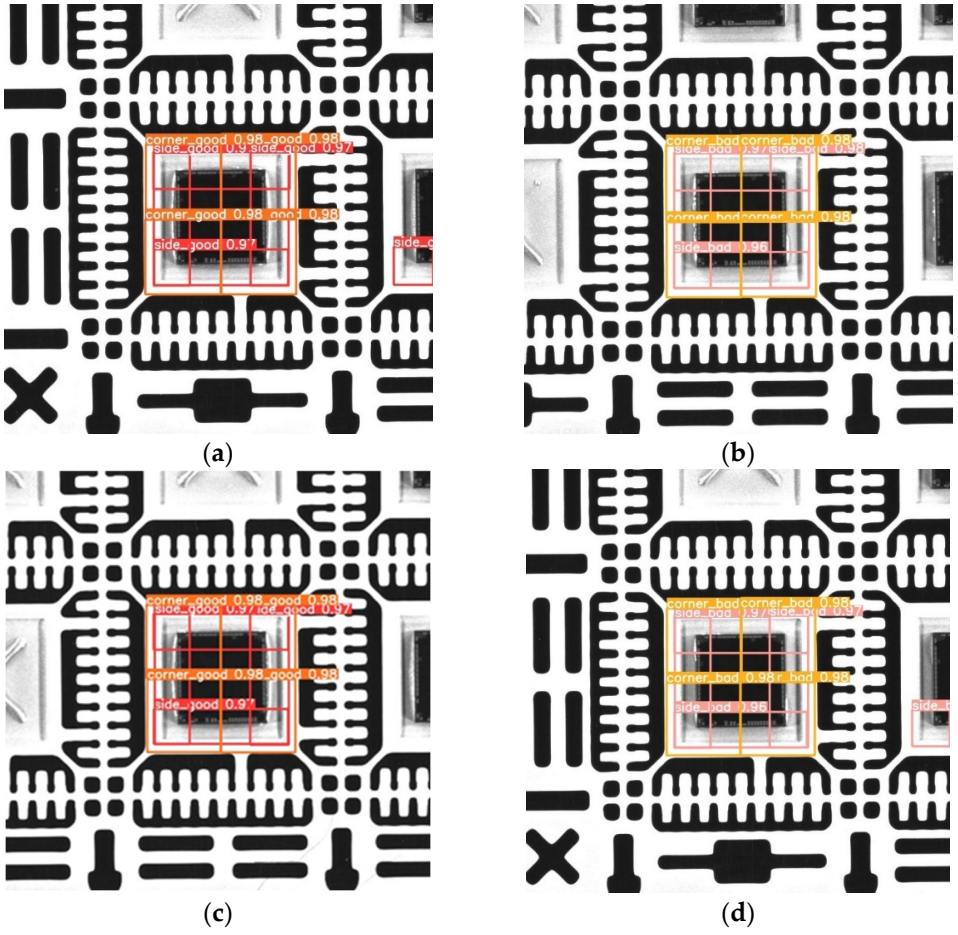
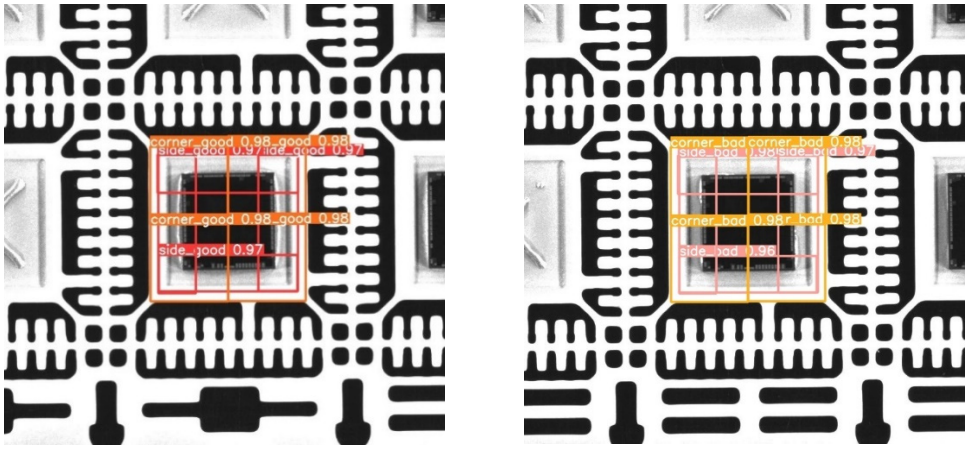


Figure 11. *Cont.*



(**e**) (**f**)

Figure 11. Die bond recognition. (**a**,**c**,**e**) bond_good; (**b**,**d**,**f**) bond_bad.

3.3. Judgment of Detection Results and Types of Die Bonding

Figure 12 illustrates the die bond detection results from classification into four output classes: side_good, side_bad, corner_good, and corner_bad. The fab defines judgment criteria. The check considered a die bond as bond_good if the bonding wholly adhered to all four die sides, all four die corners, any three die sides, or any three die corners; otherwise, it is classified as bond_bad, as shown in Figure 12.

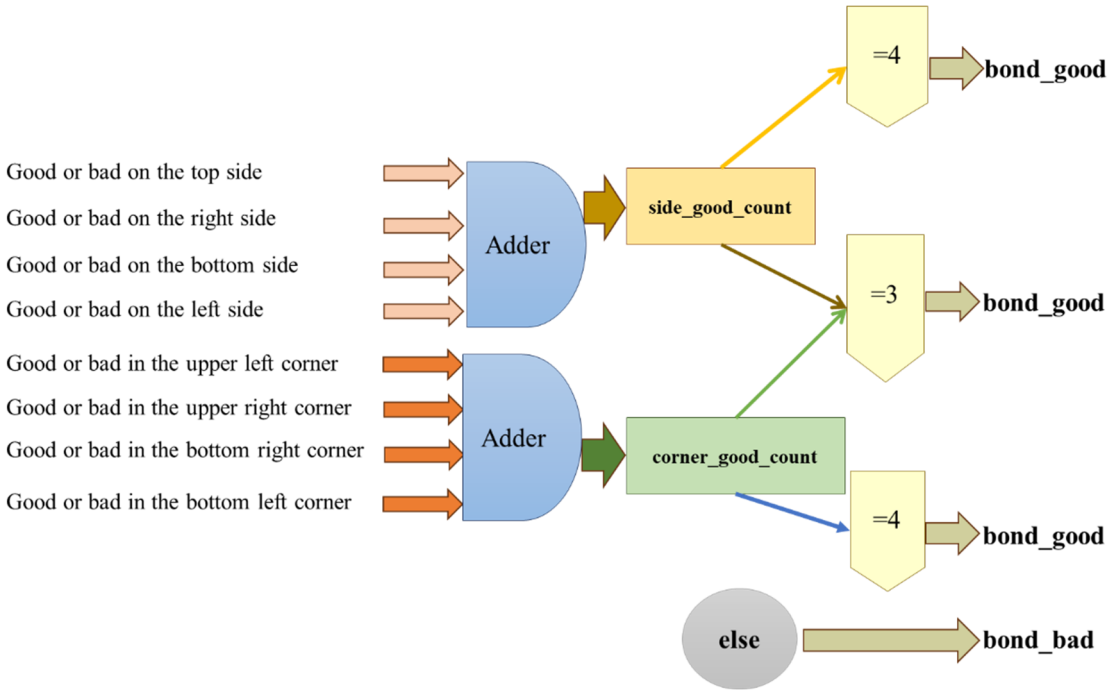


Figure 12. Judgment of categories with bond_good or bond_bad.

The die bond image is the input signal for the die bond detection model. In contrast, the output signal consists of binary outputs indicating the bonding results for the chip's sides and corners, either good or bad [19], as shown in Figure 13. In Figure 13, if the model's output signal indicates the condition of side_good or corner_good, a "1" can be assigned to the corresponding bit. Conversely, if the condition is side_bad or corner_bad, a "0" can be assigned to the corresponding bit. The model output signal for the four sides of the chip has a length of four bits, which designates the side bond code of the chip's sides. Similarly, the model output signal for the four corners of the chip also has a length of four

bits, which designates the corner bond code of the chip's corners. In other words, Figure 13 shows the arbitrary combination of the side and corner bond codes, representing the type of die bond—the combinations of the two hexadecimal values, ranging from 00x to FFx.

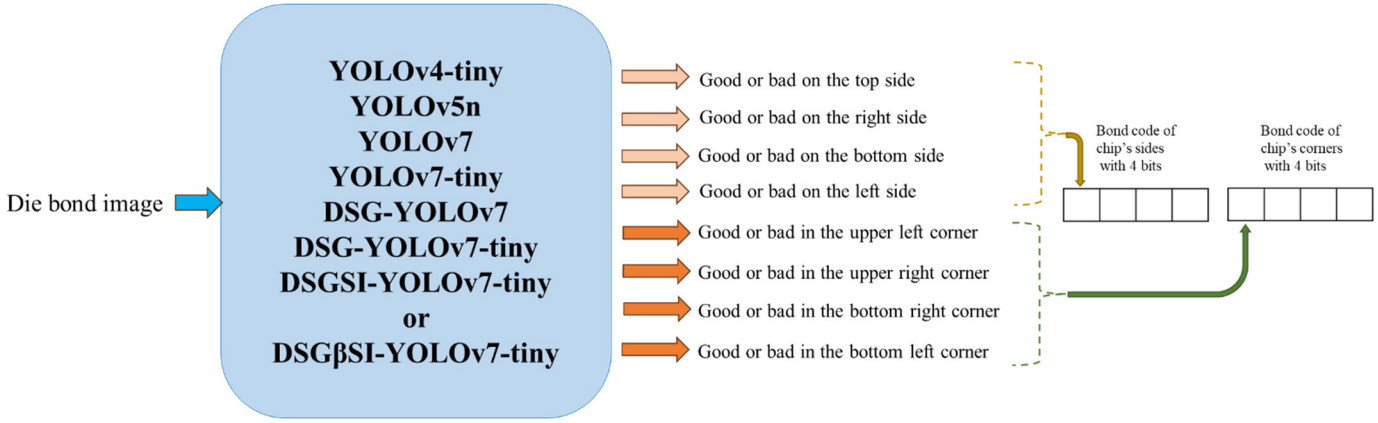


Figure 13. Any combination of the bond detection of the chip's sides and corners.

Table 1 displays the various die bond types, where the index "s" represents the bond code for the chip's sides, the index "c" stands for the bond code for the chip's corners, the symbol "b" indicates the condition of bond_bad, and the symbol "g" denotes the condition of bond_good. Each index interprets a hexadecimal value ranging from 0x to Fx for any code of the chip's sides or corners. In Table 1, we can encode the arbitrary combinations of the code of the chip's sides and corners in hexadecimal from 00x to FFx. Combining two hexadecimal codes from 00x to FFx represents the type of die bond. Statistics on the die bond type can identify the locations where bond defects predominantly occur on the chip's sides or corners. Consequently, the fab can adjust critical parameters of the machine promptly to improve the occurrence of such defects, thereby increasing the yield of the die bond process and significantly reducing manufacturing cost losses.

Table 1. The type of die bond.

c \ s	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
1	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
2	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
3	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
4	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
5	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
6	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
7	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g
8	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
9	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
A	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
B	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g
C	b	b	b	b	b	b	b	g	b	b	b	g	b	g	g	g
D	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g
E	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g
F	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g

Figure 14 shows the storage of the predicted classification results from the tests in a separate folder. Subsequently, this study will compile the actual classifications and the expected results to create a confusion matrix for calculating precision. Additionally, we can quickly assess the integrity of each chip's bonding and continuously monitor the machine's operational status. If any issues arise, the engineer can adjust the machine's parameters to prevent excessive poor chip bonding.

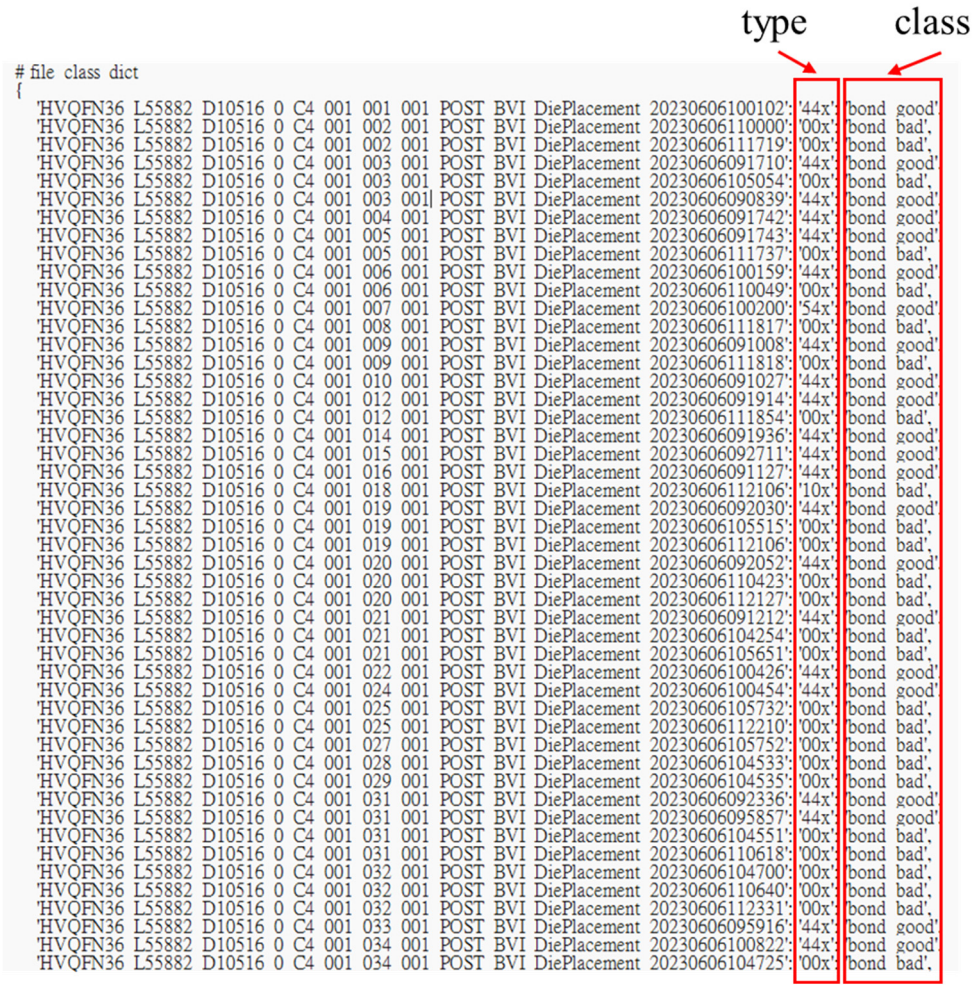


Figure 14. Prediction classification with type of die bond.

3.4. Model Improvement

Following the DSG-Yolov7 model, this study also focuses on improving the Yolov7-tiny model by ablation study using the trial and error of employing DSGConv to the different modules in the backbone, neck, and head to realize lightweight model enhancement, and this model is referred to as DSG-Yolov7-tiny, as shown in Figure 15. Figure 15 illustrates the approach of using depthwise separable convolution combined with Ghost Convolution to replace traditional convolution, achieving the goal of a lightweight model. Some designs replace specific traditional convolution layers with GhostNet, which can effectively reduce the model's parameter count and enhance overall execution performance [20]. Although GhostConv can significantly decrease computational load, using traditional convolution in specific critical locations of the model is crucial for maintaining inference precision. Therefore, in the critical nodes of the backbone and prediction sections, employing traditional convolution helps to preserve the stability of the network structure and the object precision of the inference, as shown in Figure 15.

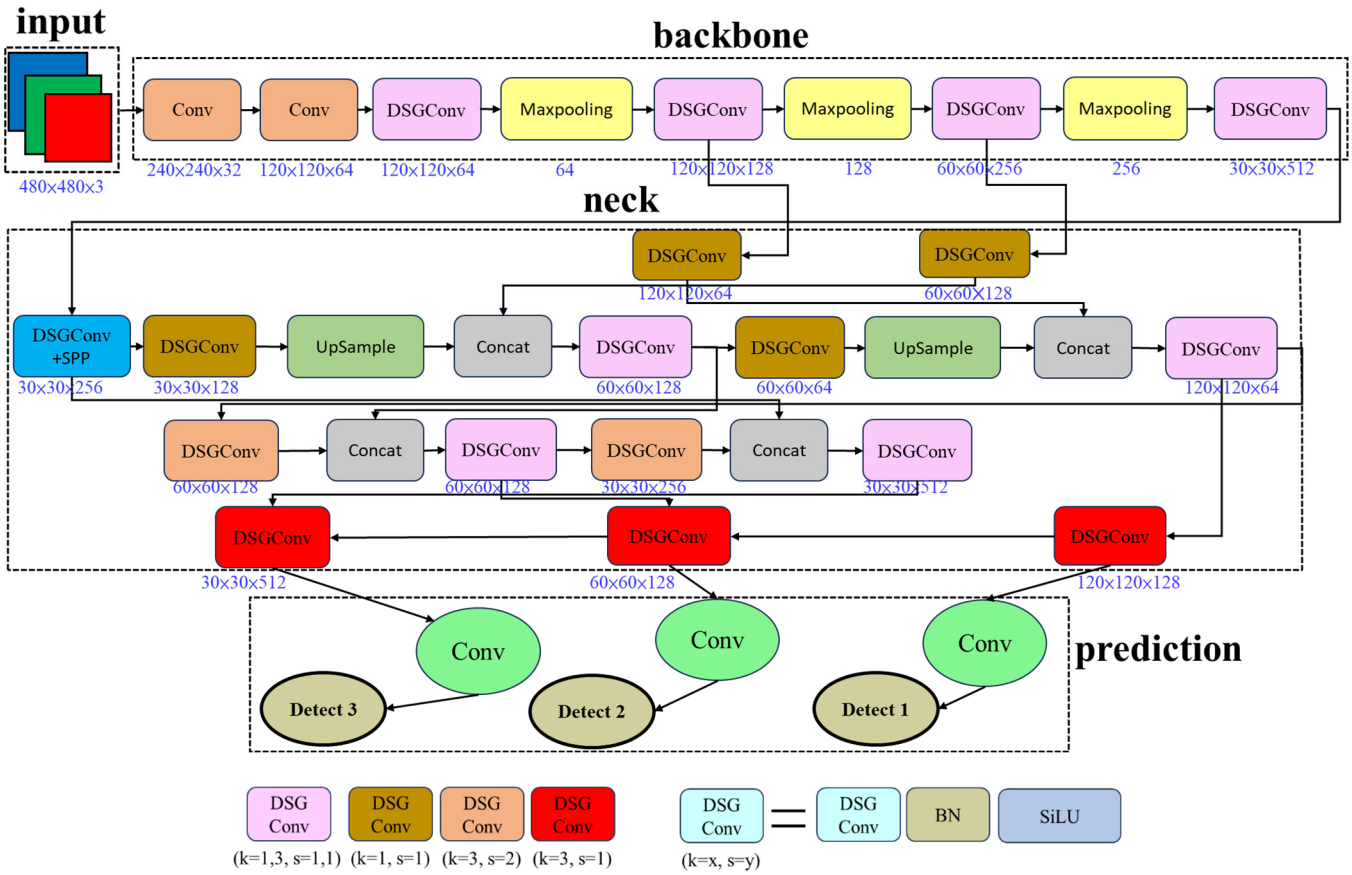


Figure 15. DSG-Yolov7-tiny architecture. Note: k represents kernel size, and s stands for stride.

However, the DSGConv architecture for lightweight models decreases the complexity of inference calculations, which may lead to a slight reduction in precision. Equation (1) computes the Rectified Linear Unit (ReLU) activation function, where x represents the input and $relu(x)$ stands for the output. In the DSG-Yolov7-tiny architecture, the activation function used is the LeakyRectified Linear Unit (LeakyReLU), an improvement of the ReLU activation function. Equation (2) calculates LeakyReLU, where x represents the input, $Lrelu(x)$ stands for the output, and α denotes the slope of the function's output when the input is negative; it is a small positive number, usually around 0.01. The ReLU function outputs zero for negative input values, causing the neuron to lose its learning ability. LeakyReLU addresses this issue by introducing a slight slope while retaining ReLU's computational simplicity and sparse activation characteristics.

$$relu(x) = \begin{cases} x & if \ \ x \geq 0 \\ 0 & if \ \ x < 0 \end{cases} \tag{1}$$

$$Lrelu(x) = \begin{cases} x & if \ \ x \geq 0 \\ \alpha x & if \ \ x < 0 \end{cases} \tag{2}$$

3.5. Model Enhancement

As mentioned in the previous section, the LeakyReLU activation function outputs with a fixed slope, so it may not optimally adapt to different data distributions or network layers. On the other hand, the introduction of negative outputs by LeakyReLU can, in some instances, affect the overall learning effectiveness of the network, leading to a decline in model performance. Improving the activation function within the architecture can enhance inference precision [21,22]. Therefore, this study modifies the activation function in the DSG-Yolov7-tiny architecture, specifically replacing LeakyReLU with the Sigmoid Linear

Unit (SiLU). SiLU is a smooth activation function that helps reduce jitter during training and fosters more stable gradient updates. Following the DSG-Yolov7 model, this study also stresses the enhancing of the Yolov7-tiny model by ablation study using the trial and error of employing SiLU, ModifiedSiLU, and AdaptiveSiLU to the different modules in the backbone, neck, and head to improve the prediction precision of the proposed model, as shown in Figure 15.

The characteristic of this activation function, which lies between linear and nonlinear, allows SiLU to be more flexible in capturing subtle changes in input data compared to other activation functions such as ReLU and LeakyReLU. Additionally, incorporating SiLU improves the overall stability of gradients, and experimental results later confirmed a slight increase in precision and precision.

SiLU, like LeakyReLU, also produces negative outputs for negative inputs. Equation (3) evaluates the Sigmoid function, where x represents the input and $\sigma(x)$ denotes the output. When x is very large, e^{-x} approaches zero, making $\sigma(x)$ approach 1. When x is very small (negative), e^{-x} becomes very large, and $\sigma(x)$ approaches 0. The Sigmoid function maps any real input x to the interval [0, 1]. Equation (4) estimates the SiLU activation function, where x represents the input, $\sigma(x)$ stands for the Sigmoid function, and $silu(x)$ is the output. Similar to LeakyReLU, SiLU produces negative outputs for negative inputs.

$$\sigma(x) = 1/\left(1 + e^{-x}\right) \tag{3}$$

$$silu(x) = x \cdot \sigma(x) \tag{4}$$

In some cases, using SiLU only resulted in slight improvements in precision, leading to the development of two variations related to SiLU: ModifiedSiLU and AdaptiveSiLU. The purpose of adopting these revised versions of SiLU is to enhance inference speed while maintaining high precision. Equation (5) calculates ModifiedSiLU, where β represents a learnable parameter and $Mosilu(x)$ stands for the output. ModifiedSiLU introduces an adjustable β parameter to control the steepness of the curve. When x is large, $\sigma(\beta x)$ approaches 1, and when x is small, $\sigma(\beta x)$ approaches 0. The initial β value is 1.0, which limits the flexibility of function adjustment, resulting in a fixed curve slope. Therefore, β is adjusted to optimize the activation effect in different layers, first increasing the value to 1.5 to achieve more robust nonlinear feature extraction. Then, β is gradually reduced to 1.25 for a flatter activation curve, resulting in better gradient flow.

$$Mosilu(x) = x \cdot \sigma(\beta x) \tag{5}$$

$\beta_a(x)$ is a neural network, and Equation (6) computes $\beta_a(x)$, where x represents the input. The initial value of the first layer's bias b_1 is set to 0, and the initial value of the output layer's bias b_2 is also set to 0. The initial values of the weight matrix W_1 for the first layer and W_2 for the output layer are randomly generated. $\beta_a(x)$ behaves similarly to a fixed β learning mechanism, providing greater flexibility and adaptability, enabling it to handle input-related scaling. Equation (7) calculates AdaptiveSiLU, where x represents the input, and the initial value of the bias b is -1. When b = -1, the function behaves more like ReLU near x = 0, causing $\sigma(\beta_a(x) \cdot x + b) \approx 0$.

$$\beta_a(x) = W_2 \cdot relu(W_1 x + b_1) + b_2 \tag{6}$$

$$Apsilu(x) = x \cdot \sigma(\beta_a(x) \cdot x + b) \tag{7}$$

According to Xavier [23,24], Equation (8) evaluates the initial values of W_1 and W_2 as uniformly distributed random numbers, where W represents the initialized weight matrix, and $U(a, b)$ stands for the uniform distribution ranging from a to b. n_i indicates the input dimension of the layer, which will be the size of the input data 480 × 480, and n_{out} represents the output dimension of the layer, corresponding to the number of classes n_{out} = 4. The term $\sqrt{6}$ is the coefficient used in Xavier initialization to ensure the weights

maintain a stable variance during forward propagation. After initializing the weight matrixes W_1 and W_2, weight updates using backpropagation can change their values. First, the network makes predictions and computes the error (loss). Then, backpropagation calculates the loss gradient to the weights. Finally, the optimizer (such as gradient descent) updates W_1 and W_2 by adjusting them in a direction that reduces the error. The training repeated this process to gradually enhance the performance of the network. Therefore, the input dimension of W_1 is the output dimension of the previous layer (n_i), and the output dimension is the number of neurons in the hidden layer (n_{out}). The input dimension of W_2 is the output dimension of the hidden layer, and the output is one because it represents the scaling factor $\beta_a(x)$.

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{(n_i + n_{out})}}, \frac{\sqrt{6}}{\sqrt{(n_i + n_{out})}}\right) \tag{8}$$

In Figure 16, the DSGβSI-Yolov7-tiny architecture is modified primarily in the backbone section. The network avoids setting all β values to 1.0, which would fix the slope of the curve and reduce the flexibility of function adjustment; it uses ModifiedSiLU in the early layers (0, 1, 5, and 10), with β starting at 1.5 and decreasing to 1.25 after two layers. This approach allows the model to achieve better gradient flow. The later layers maintain the original SiLU to ensure gradient stability, helping the model quickly extract prominent features while balancing feature extraction and information transfer.

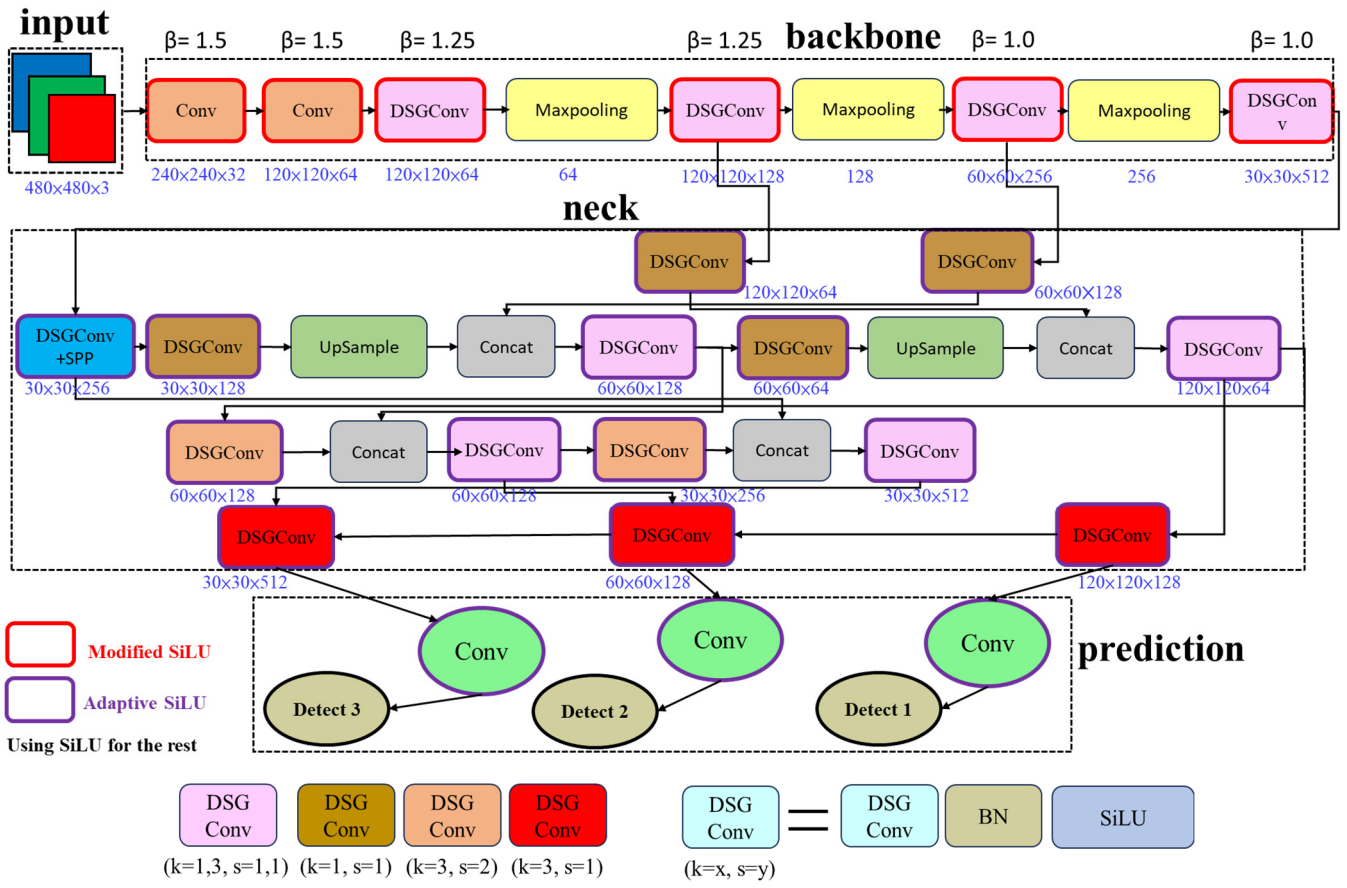


Figure 16. DSGβSI-Yolov7-tiny architecture. Note: k represents kernel size, and s stands for stride.

In the neck section, the network employs AdaptiveSiLU to enable the model to handle complex feature combinations in the middle part of the network. Finally, the network utilizes AdaptiveSiLU again in the head (prediction) section, allowing the model to autonomously determine the optimal activation function.

3.6. Build a Model

This study trained eight models, Yolov4-tiny, Yolov5n, Yolov7, Yolov7-tiny, DSG-Yolov7, DSG-Yolov7-tiny, DSGSI-Yolov7-tiny, and DSGβSI-Yolov7-tiny, for predictions. Each model randomly initialized the parameters, and during the training phase, the parameters were gradually adjusted through trial and error, as shown in Table 2. Table 2 presents the optimized parameter settings for each model in this experimental case. Figure 13 illustrates how the models take images as input signals, using detection and prediction to determine the bonding conditions of each die. The output signals indicate the die bond quality and the die bond type, which can display the bonding status of the four die's sides and four die's corners.

Table 2. Hyperparameter settings.

Model	Hyperparameters
Yolov4-tiny	ep = 120, bs = 16, is = 1024 × 1024, op = SGD, act = Leaky ReLU, Sigmoid, tc = $loss \leq 10^{-3}$
Yolov5n	ep = 120, bs = 16, is = 1024 × 1024, op = Adam, act = SiLU, Sigmoid, tc = $loss \leq 10^{-3}$
Yolov7	ep = 120, bs = 16, is = 1024 × 1024, op = AdamW, act = Leaky ReLU, Sigmoid, tc = $loss \leq 10^{-3}$
Yolov7-tiny	ep = 120, bs= 16, is = 1024 × 1024, op = AdamW, act = Leaky ReLU, Sigmoid, tc = $loss \leq 10^{-3}$
DSG-Yolov7	ep = 120, bs= 16, is = 1024 × 1024, op = AdamW, act = Leaky ReLU, Sigmoid, tc = $loss \leq 10^{-3}$
DSG-Yolov7-tiny	ep = 120, bs= 16, is = 1024 × 1024, op = AdamW, act = Leaky ReLU, Sigmoid, tc = $loss \leq 10^{-3}$
DSGSI-Yolov7-tiny	ep = 120, bs= 16, is = 1024 × 1024, op = AdamW, act = SiLU, Sigmoid, tc = $loss \leq 10^{-3}$
DSGβSI-Yolov7-tiny	ep = 120, bs= 16, is = 1024 × 1024, op = AdamW, act = ModifiedSiLU, AdaptiveSiLU, SiLU, Sigmoid, tc = $loss \leq 10^{-3}$

Note: ep represents epoch, bs stands for batch size, is indicates image size, op means optimizer, ap denotes activation function, and tc is termination condition.

3.7. The Workflow of the System

First, the system performs data preprocessing on the collected 3145 images provided by the semiconductor manufacturer. Data preprocessing involved organizing the data, using LabelImg to label each image, and converting the XML files into the VOC file format, which YOLO accepts. Next, the dataset was divided into training, validation, and testing sets, with approximate proportions of 70%, 15%, and 15%, respectively. During the training phase, modeling sets the parameter epoch to 120, batch size to 16, and input image size to 1024 × 1024. The models selected for training include Yolov4-tiny, Yolov5n, Yolov7, and Yolov7-tiny, along with the modified models DSG-Yolov7, DSG-Yolov7-tiny, DSGSI-Yolov7-tiny, and DSGβSI-Yolov7-tiny. After the training, the testing phase involves inference and recognition using the test set. Finally, we examine each model's training and testing results to evaluate their performance, as shown in Figure 17.

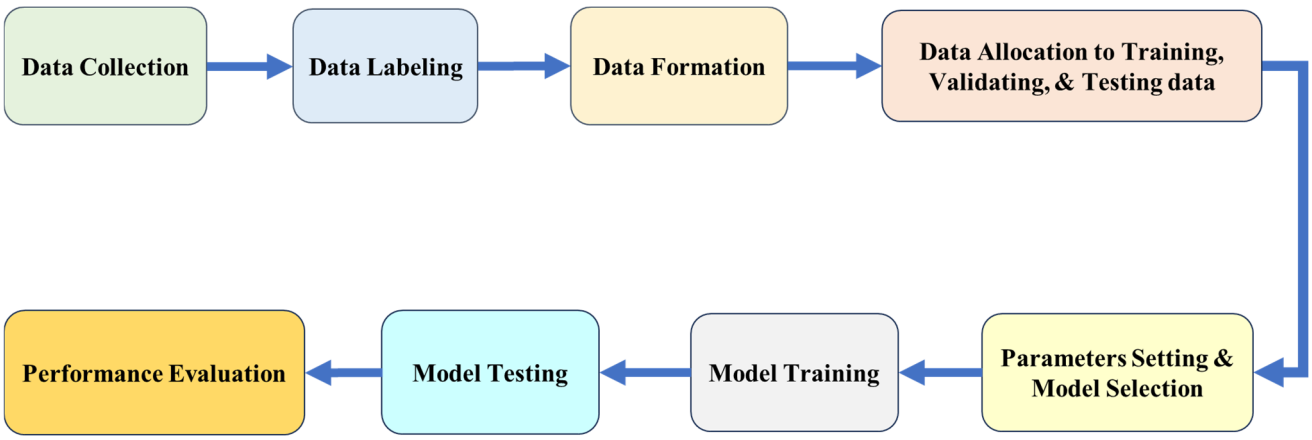


Figure 17. The workflow of the system.

4. Experimental Results and Discussion

4.1. Experimental Environment

Table 3 displays the hardware configuration used in this experiment. Table 4 outlines the software packages utilized in the experiment. Data preprocessing involved the use of LabelImg to label each image. The next step was to use Jupyter Notebook to convert the XML files into text files and divide the dataset. Then, the experiment used Anaconda Prompt, Python, and PyTorch to execute training and recognition. The experiment utilized TensorFlow to monitor the training progress. Finally, based on the recognition results for each image, this study categorized chips with complete bonding as bond_good and those with incomplete bonding as bond_bad using Jupyter Notebook. Furthermore, this study evaluated the performance of multiple models and used the Anaconda Prompt to export the best model in TensorFlow Lite format for use in production line machinery.

Table 3. Hardware specifications.

Resource	Workstation
GPU	NVIDIA GeForce RTX 4070 Ti
CPU	Intel(R) Xeon(R) W-2223 CPU @ 3.60 GHz
Memory	32 GB
Storage	1 TB × 1 (HDD)
Jetson Nano	NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores

Table 4. List of packages.

Software	Version
LabelImg	1.8
Anaconda® Individual Edition	4.9.2
Jupyter Notebook	6.1.4
TensorFlow	v2.14.0
PyTorch	1.6
Python	3.6.9

4.2. Data Collection and Model Evaluation

We primarily used Anaconda 3 to train the Yolov5 series, Yolov4-tiny, Yolov7, and Yolov7-tiny on a PC and then deployed them on a Jetson Nano embedded system to compare the results of each model. Next, we also deployed the improved models on the Jetson Nano for a comprehensive performance evaluation of all models. The data source comprised 3145 images from a well-known semiconductor manufacturer in southern Taiwan, where the first dataset collected 2491 training images from machine #1, and the second dataset 474 test images from machine #2. This experiment uses the data arrangement to train and test the model to achieve the effect of cross-dataset evaluation. This study annotated each image using the LabelImg software, followed by data preprocessing. The data preprocess divided the training dataset into 2204 images (approximately 70%) for training and 467 images (approximately 15%) for validation. The experiment tested various object detection models on a workstation platform. During the training and validation process, this study adopted k-fold cross-validation [25], selecting k = 5 to evaluate the Yolo-related modeling comprehensively. Modeling recorded the training time for the same set of training data on the workstation, and using Equation (9), we calculated the total inference time IT_i required for various object detection models on 474 test images (approximately 15%). In Equation (9), i represents the i-th object detection model used for image inference, I stands for the total number of object detection models, x denotes the x-th test image, X indicates the total number of test images, and EIT_i is the time taken to complete the inference for each test image.

$$IT_i = \sum_{x=1}^{X} EIT_i, \ \ where \ \ i = 1, \ 2, \ \ldots, \ I, \quad x = 1, \ 2, \ldots, \ X \tag{9}$$

The experimental setup defined the test image size as 1024 × 1024, with a batch size set to 16 and the number of iterations set to 120. In Table 5, the first row shows the time required for training different object detection models based on the same parameter settings. The second row calculates the time to infer 474 images within the same test set. The experimental results indicate that the proposed DSGβSI-Yolov7-tiny model has a shorter inference time than the other models.

Table 5. Training and inference times.

Phase	Yolov4-Tiny	Yolov5n	Yolov7	Yolov7-Tiny	DSG-Yolov7	DSG-Yolov7-Tiny	DSGSI-Yolov7-Tiny	DSGβSI-Yolov7-Tiny
Training (h)	30.1	14.9	58.8	46.9	56.4	3.6	3.6	4.1
Inference (ms)	25.1	18.6	19.3	18.2	16.3	5.4	5.7	5.2

Table 6 lists each object detection model's parameters and FLOPs (Gflops). In Table 6, before applying DSGConv for making the models lightweight, the Yolov4-tiny model had the highest number of parameters, while the Yolov5n model had the fewest. After the DSG modifications, the models DSG-Yolov7, DSG-Yolov7-tiny, DSGSI-Yolov7-tiny, and DSGβSI-Yolov7-tiny all showed a significant reduction in the number of parameters, achieving the goal of being lightweight.

Table 6. Parameters and flop.

Feature	Yolov4-Tiny	Yolov5n	Yolov7	Yolov7-Tiny	DSG-Yolov7	DSG-Yolov7-Tiny	DSGSI-Yolov7-Tiny	DSGβSI-Yolov7-Tiny
Parameter (#)	6,056,610	1,764,577	60,231,067	36,497,954	22,363,042	605,242	745,378	743,829
Flop (Gflops)	14.0	4.1	13.2	103.2	61.3	1.6	1.9	1.8

4.3. Experimental Results

In Figure 18, the PR curve plots recall on the X-axis and precision on the Y-axis, with each point representing different threshold values leading to varying recall and precision results for the mean Average Precision (mAP) calculation. The mAP result is obtained by summing the AP values calculated from recall and precision across all classes and dividing by the total number of classes. Before applying DSGConv for lightweight Yolo-related models, the Yolov7-tiny achieved the best precision, while Yolov4-tiny had the lowest precision. After the DSG modifications, both DSG-Yolov7 and DSG-Yolov7-tiny showed improved speed, but the omission of some complex calculations led to a slight decrease in precision. However, with further refinement in the DSGSI-Yolov7-tiny model, there was a slight improvement in model precision, and an additional enhancement in the DSGβSI-Yolov7-tiny model resulted in even higher precision.

After the model training is complete, we can use visualization tools to present the results of the loss plot. Under a batch size setting of 16, the loss plot for the DSGβSI-Yolov7-tiny model after 120 training epochs is shown in Figure 19. During the 120 training epochs, this study produced six plots to display the loss curves of the training precision. In Figure 19, the first, second, and third rows represent the localization loss, confidence loss, and matching loss between predictions and ground truth. The first and second columns also represent the training loss and validation loss. In the second column, the abbreviation "val" indicates validation.

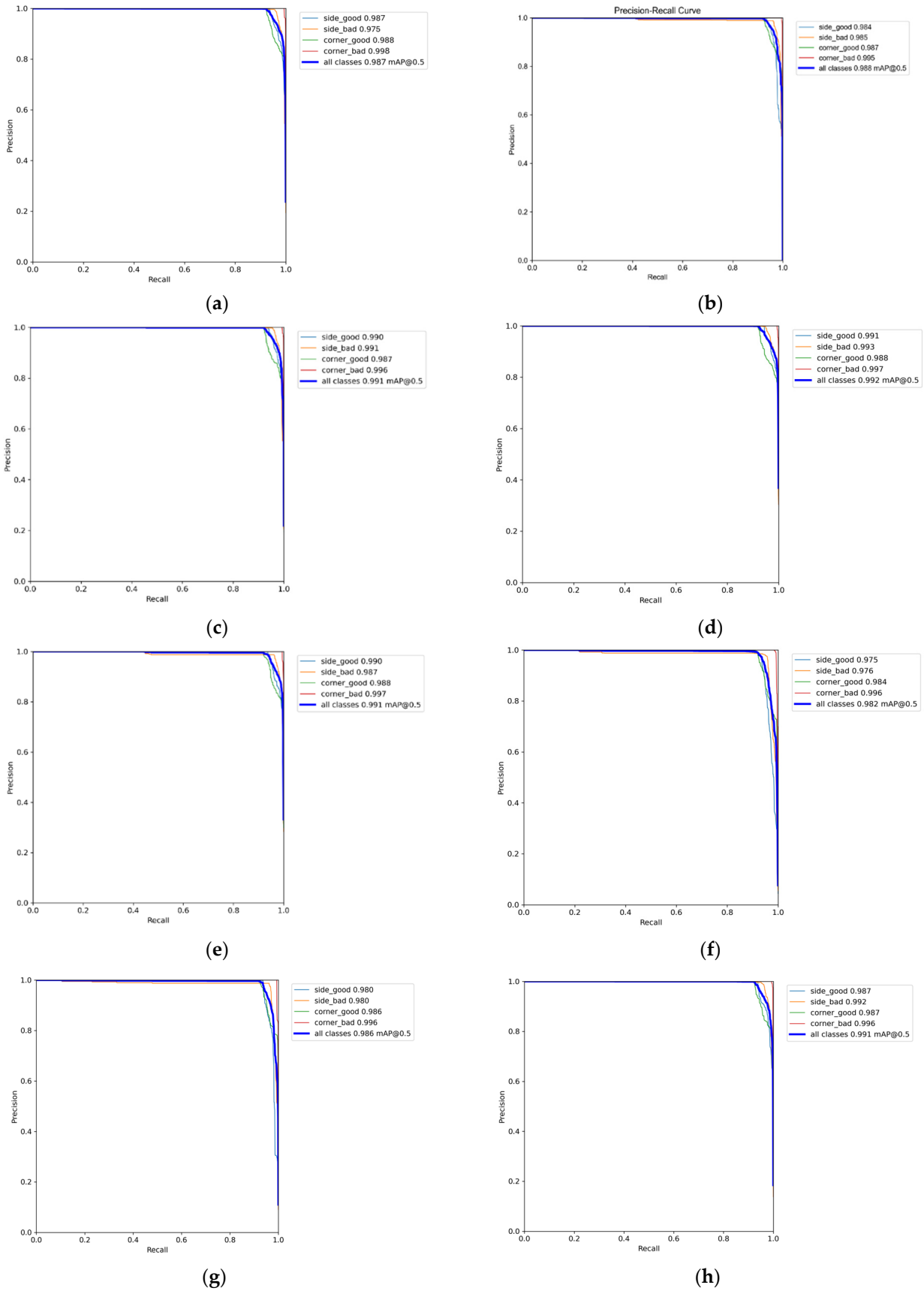


Figure 18. The precision–recall curve for the object detection model. (**a**) Yolov4-tiny; (**b**) Yolov5n; (**c**) Yolov7; (**d**) Yolov7-tiny; (**e**) DSG-Yolov7; (**f**) DSG-Yolov7-tiny; (**g**) DSGSI-Yolov7-tiny; (**h**) DSGβSI-Yolov7-tiny.

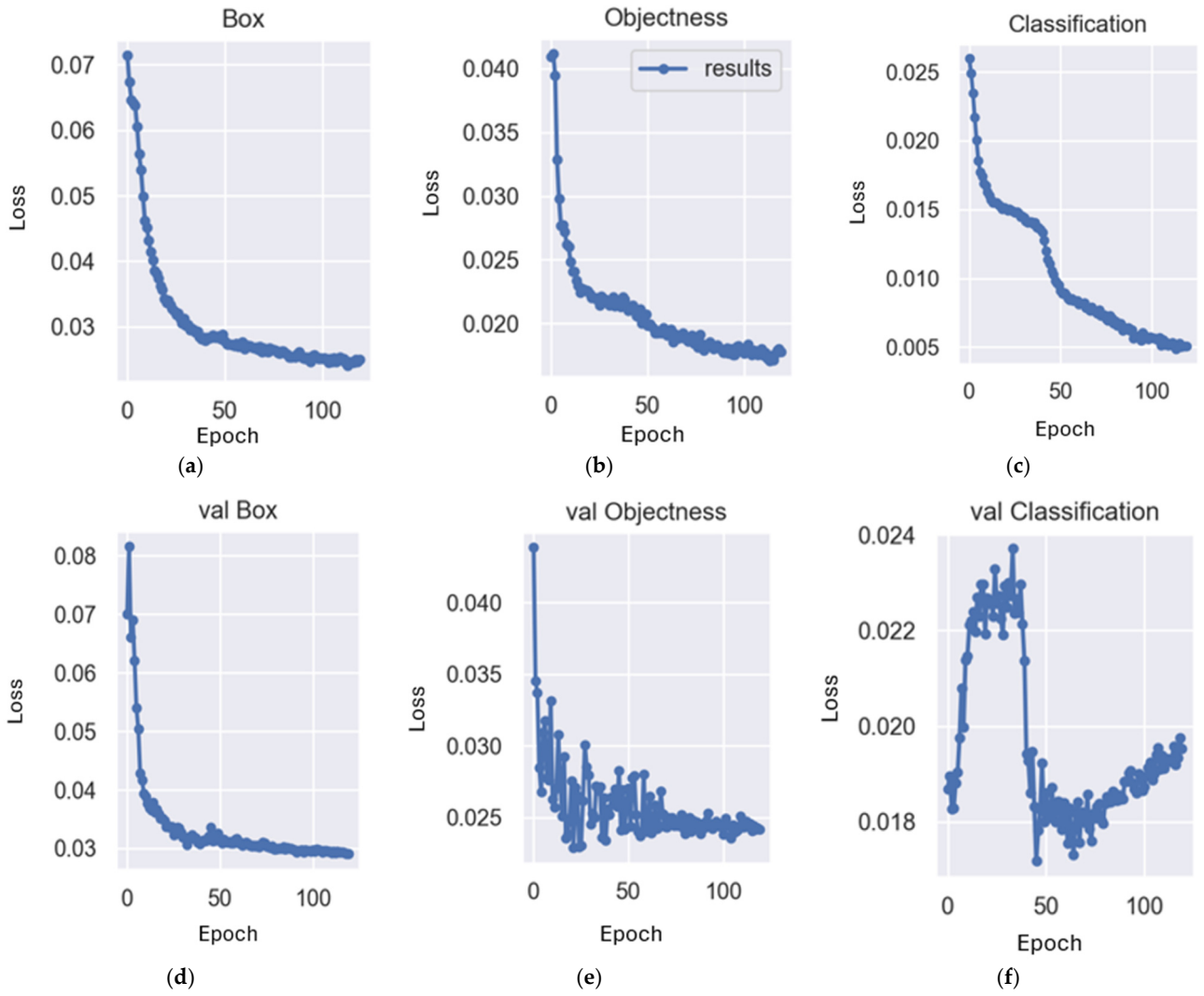


Figure 19. Loss plot of DSGβSI-Yolov7-tiny. (**a**) Box loss; (**b**) objectivity loss; (**c**) classified losses; (**d**) verify box loss; (**e**) verify objectivity loss; (**f**) verify classification losses.

4.4. Performance Evaluation

Equation (10) calculates frames per second (FPS) to show the execution speed of object detection. According to Equation (10), it computes FPS_j, the FPS obtained from different object detection models, where $IRAIT_j$ represents the time required for each image in real-time video input using the different object detection models, J stands for the total number of object detection models, and j denotes the calculation for the j-th object detection model.

$$FPS_j = \frac{1}{IRAIT_j}, \ \ where\ j = 1,\ 2, \ldots,\ J \tag{10}$$

Equation (11) evaluates object detection precision using mean Average Precision (mAP), calculated by finding the average precision for each category and then computing the overall average. Equation (11) estimates the mAP_l for different object detection models, where l represents the l object detection model used for calculating mAP_l, L stands for the total number of object detection models, C_l indicates the number of categories that a specific model needs to identify, k_l denotes a specific category of the model, and AP_{k_l} refers to the precision for that specific category within the model.

$$mAP_l = \frac{\sum_{k_l=1}^{C_l} AP_{k_l}}{C_l}, \ \ where\ k_l = 1,\ 2,\ \ldots,\ C_l,\ l = 1, 2, \ldots, L \tag{11}$$

Next, we evaluate the execution speed and precision of different object detection models. After training various object detection models with the same parameters, we tested them using a set of 474 images and plotted the results of the PR curve. Based on Equations (10) and (11), we calculated the execution speed (FPS) and precision (mAP) from the tests, as shown in Table 7.

Table 7. Performance indexes.

Metric	Yolov4-Tiny	Yolov5n	Yolov7	Yolov7-Tiny	DSG-Yolov7	DSG-Yolov7-Tiny	DSGSI-Yolov7-Tiny	DSGβSI-Yolov7-Tiny
FPS	39.8	53.8	51.8	54.9	61.3	185.2	175.4	192.3
Precision	98.5	98.8	99.0	99.1	99.1	98.2	98.6	99.1
Recall	95.2	95.1	95.2	95.0	95.3	95.1	95.3	95.4
F1-score	0.96	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Accuracy	98.7	98.9	98.9	98.5	99.0	98.0	98.5	98.5

Note: the precision, recall, and accuracy values are given as percentages.

Table 7 compares the performance of different versions of Yolo-related models, including Yolov4-tiny, Yolov5n, Yolov7, and Yolov7-tiny, as well as the lightweight models DSG-Yolov7, DSG-Yolov7-tiny, DSGSI-Yolov7-tiny, and DSGβSI-Yolov7-tiny. The performance evaluation metrics include FPS, precision, recall, F1-score, and accuracy. Table 7 also displays the performance of these models across these metrics. The experimental results indicate that the DSGβSI-Yolov7-tiny model achieved the best FPS, precision, recall, F1-score, and accuracy performance.

4.5. Discussion

First, regarding speed metrics (FPS), the Yolov7-tiny model achieved a fast inference speed of 54.9 FPS. However, the inference speeds of the other four models—DSG-Yolov7, DSG-Yolov7-tiny, DSGSI-Yolov7-tiny, and DSGβSI-Yolov7-tiny—were significantly higher than that of the Yolov7-tiny model, with ratios of 1.12, 3.25, 3.19, and 3.5, respectively. These results indicate that the DSG approach can significantly enhance inference speed. Secondly, regarding the precision metric, the results for the four models mentioned above were almost identical. The above suggests that the DSG approach did not significantly alter the precision levels.

Additionally, regarding the accuracy metric, there were only slight differences among the four models. The above indicates that the DSG method does not negatively impact image recognition accuracy. Thus, the lightweight improvements made through DSGCONV significantly enhance the inference speed of the models while maintaining a consistent level of prediction precision and image recognition accuracy.

The DSGSI-Yolov7-tiny model, which employs the SIReLU activation function, achieved the fastest inference speed, considerably boosting overall performance. However, to ensure that the model maintains precision above 99%, the DSGβSI-Yolov7-tiny model was developed, achieving optimal yield rates and reducing loss costs in the die bond process.

Hsu et al. [10] mention depthwise separable convolution layers, which can increase the model's computation speed and achieve better prediction results while maintaining a smaller model size. Additionally, Zhang et al. [11] highlight the challenges of deploying convolutional neural networks (CNNs) on embedded devices with limited memory and computational resources. Therefore, combining the smaller neural networks of Yolov5 and GhostNet is more suitable for deployment on FPGAs and other memory-constrained embedded devices. This experiment integrates the Yolo-related models with GhostNet to replace traditional convolution, eliminating redundant blocks in the original architecture or replacing them with improved convolution layers. This approach significantly reduces convolution calculations, leading to a marked increase in computing speed while sustaining high predictive precision.

Lang et al. [13] pointed out that the Yolov7-tiny architecture is streamlined from the original Yolov7's backbone, neck, and head to achieve a lighter design. This experiment

realizes a more lightweight DSG-Yolov7-tiny, making the model more suitable for operation on resource-constrained devices and highlighting the significance of Yolov7-tiny. Both Sun et al. [12] and Lang et al. [13] mentioned that combining Yolov7 or Yolov7-tiny with the Ghost module allows the improved Yolov7-Ghost to maintain the original Yolov7 detection precision while also increasing inference speed, which is even more pronounced in Yolov7-tiny. This experiment implements DSG-Yolov7-tiny following the same approach to significantly enhance execution speed. Yang et al. [14] proposed improving the activation function of Yolov7-tiny to FReLU, explaining its benefits. This experiment introduces the SiLU activation function to enhance the precision of the DSG-Yolov7-tiny model, resulting in DSGSI-Yolov7-tiny. Furthermore, to improve model precision, DSGβSI-Yolov7-tiny was introduced for real-time and efficient application in die bond production machines.

This experiment also has some limitations. A single image containing multiple die bonds may lead to variations in die bond types. During the training phase, the model may not effectively learn the features of each die bond type, affecting inference precision. The best approach would be to cut a single image with multiple die bonds into separate die bond images for independent inference, but this is time-consuming. Furthermore, although the improved DSGSI-Yolov7-tiny model is swift, there is a slight decline in precision and accuracy. Maintaining the same speed as the original Yolov7-tiny model may impact these metrics.

Additionally, the performance of the GPU is a crucial factor influencing precision, accuracy, and speed. Currently, the GPU model used is the NVIDIA GeForce RTX 4070 Ti. Upgrading to a higher model, such as the NVIDIA RTX 4090, which has 24 GB of GDDR6X video memory, would enable it to handle larger deep learning models, making it particularly suitable for high utilization and throughput tasks in rapid die bond detection and prediction.

5. Conclusions

The DSGβSI-Yolov7-tiny model proposed in this study achieves the highest performance metrics for real-time and efficient prediction, making it an optimal solution for die bond detection and prediction applications. Fab can rapidly deploy our method on factory production machines for die bond detection and prediction. By analyzing the model's judgments on the classification of die bond images, we can identify the most frequently occurring defects and subsequently adjust critical parameters of the machines in real-time. This process enhances the yield of the die bond manufacturing process while significantly reducing manufacturing cost losses. For semiconductor manufacturers, this increases production yield and minimizes production losses.

Future work will extend the proposed approach to other applications, such as chip contour detection, street view analysis, mask-wearing detection, operator attire compliance, and product detection on factory production lines. Moreover, we will continue to seek a better prediction model, for example, the Yolov11n model, to replace or modify the DSGβSI-Yolov7-tiny architecture. This improvement will allow for optimizing or enhancing our proposed model, facilitating the development of more efficient die bond detection and prediction methods.

Author Contributions: B.R.C. and W.-S.C. conceived and designed the experiments; H.-F.T. collected the dataset and proofread the manuscript; and B.R.C. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: The Ministry of Science and Technology Council fully supports this work in Taiwan, Republic of China, under grant numbers NSTC 113-2622-E-390-003 and NSTC 113-2221-E-390-015.

Data Availability Statement: The Sample Programs used to support the findings of this study can be found at the following link: https://drive.google.com/file/d/1--JXy7tgrUj0fjAphe3dsi5tTjs0Ee6j/view?usp=sharing (accessed on 22 October 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, M.-F.; Chen, C.-W.; Chen, C.-Y.; Hwang, C.-H.; Hwang, L.-Y. An AOI system development for inspecting defects on 6 surfaces of chips. In Proceedings of the 2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings, Taipei, Taiwan, 23–26 May 2016; pp. 1–6.
2. Yang, Y.; Wang, X. An improved YOLOv7-tiny-based lightweight network for the identification of fish species. In Proceedings of the 2023 5th International Conference on Robotics and Computer Vision (ICRCV), Nanjing, China, 15–17 September 2023; pp. 188–192.
3. Chang, B.R.; Tsai, H.-F.; Chang, F.-Y. Applying advanced lightweight architecture DSGSE-Yolov5 to rapid chip contour detection. *Electronics* **2024**, *13*, 10. [CrossRef]
4. Chang, B.R.; Tsai, H.-F.; Chang, F.-Y. Boosting the response of object detection and steering angle prediction for self-driving control. *Electronics* **2023**, *12*, 4281. [CrossRef]
5. Li, C.; Tan, G.; Wu, C.; Li, M. YOLOv7-VD: An algorithm for vehicle detection in complex environments. In Proceedings of the 2023 4th International Conference on Computer, Big Data and Artificial Intelligence (ICCBD+AI), Guiyang, China, 15–17 December 2023; pp. 743–747.
6. Alam, L.; Kehtarnavaz, N. A survey of detection methods for die attachment and wire bonding defects in integrated circuit manufacturing. *IEEE Access* **2022**, *10*, 83826–83840. [CrossRef]
7. Phan, Q.-H.; Nguyen, V.-T.; Lien, C.-H.; Duong, T.-P.; Hou, M.T.-K.; Le, N.-B. Classification of tomato fruit using YOLOv5 and convolutional neural network models. *Plants* **2023**, *12*, 790. [CrossRef] [PubMed]
8. Hsiao, S.-F.; Tsai, B.-C. Efficient computation of depthwise separable convolution in MobileNet deep neural network models. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Penghu, Taiwan, 15–17 September 2021; pp. 1–2.
9. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
10. Wang, T.; Ray, N. Compact Depth-Wise Separable Precise Network for Depth Completion. *IEEE Access* **2023**, *11*, 72679–72688. [CrossRef]
11. Zhang, Y.; Cai, W.; Fan, S.; Song, R.; Jin, J. Object detection based on YOLOv5 and GhostNet for orchard pests. *Information* **2022**, *13*, 548. [CrossRef]
12. Sun, D.; Zhang, L.; Wang, J.; Liu, X.; Wang, Z.; Hui, Z.; Wang, J. Efficient and accurate detection of herd pigs based on Ghost-YOLOv7-SIoU. *Neural Comput. Appl.* **2024**, *36*, 2339–2352. [CrossRef]
13. Lang, C.; Yu, X.; Rong, X. LSDNet: A lightweight ship detection network with improved YOLOv7. *J. Real-Time Image Process.* **2024**, *21*, 60. [CrossRef]
14. Wu, Y.; Tang, Y.; Yang, T. An improved nighttime people and vehicle detection algorithm based on YOLOv7. In Proceedings of the 2023 3rd International Conference on Neural Networks, Information and Communication Engineering (NNICE), Guangzhou, China, 24–26 February 2023; pp. 266–270.
15. Shafiee Sarvestani, A.; Zhou, W.; Wang, Z. Perceptual Crack Detection for Rendered 3D Textured Meshes. *arXiv* **2024**, arXiv:2405.06143.
16. Song, X.; Chen, L.; Zhang, L.; Luo, J. Optimal proposal learning for deployable end-to-end pedestrian detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 3250–3260.
17. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. GhostNet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1580–1589.
18. Madhuri, P.; Akhter, N.; Raj, A.B. Digital implementation of depthwise separable convolution network for AI applications. In Proceedings of the 2023 IEEE Pune Section International Conference (PuneCon), Pune, India, 14–16 December 2023; pp. 1–5.
19. Lee, J.H.; Kong, J.; Munir, A. Arithmetic coding-based 5-bit weight encoding and hardware decoder for CNN inference in edge devices. *IEEE Access* **2021**, *9*, 166736–166749. [CrossRef]
20. Mou, C.; Zhu, C.; Liu, T.; Cui, X. A novel efficient wildlife detecting method with lightweight deployment on UAVs based on YOLOv7. *IET Image Process.* **2024**, *18*, 1296–1314. [CrossRef]
21. Li, B.; Liu, L.; Wang, S.; Liu, X. Research on object detection algorithm based on improved YOLOv7. In Proceedings of the 9th International Conference on Computer and Communication (ICCC), Chengdu, China, 8–11 December 2023; pp. 1724–1727.
22. Sun, H.-R.; Shi, B.-J.; Zhou, Y.-T.; Chen, J.-H.; Hu, Y.-L. A Smoke Detection Algorithm Based on Improved YOLO v7 Lightweight Model for UAV Optical Sensors. *IEEE Sens. J.* **2024**, *24*, 26136–26147. [CrossRef]
23. Sai, T.A.; Lee, H. Weight initialization on neural network for neuro pid controller-case study. In Proceedings of the 2018 International Conference on Information and Communication Technology Robotics (ICT-ROBOT), Busan, Republic of Korea, 6–8 September 2018; pp. 1–4.

24. Wong, K.; Dornberger, R.; Hanne, T. An analysis of weight initialization methods in connection with different activation functions for feedforward neural networks. *Evol. Intell.* **2024**, *17*, 2081–2089. [CrossRef]
25. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv* **2015**, arXiv:1502.03167.