Supplementary

This supporting information is included for further reference for the reader. The supplementary materials document contains diagrams of the network structure and experimental results that provide additional illustrations to the article.

Architecture of EfficientNetV2's blocks

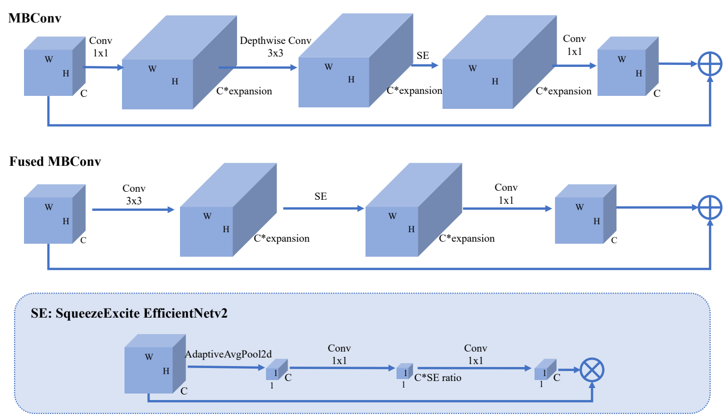


Figure 1: The architecture of Fused MBConv and naive MBConv.

As Fig. 1 indicates, MBConv starts with a 1x1 convolutional layer for upscaling, followed by a BN and Swish activation function; then the following is a depthwise convolutional layer, and its output is fed to the SE module. The SE module begins with a pooling layer shaping the input into 1x1xC dimensions, next the result is downscaled, upscaled, and multiplied with the original input; finally, the output is obtained by dimensionality reduction via 1x1 convolutional layer with shortcut branch summation. In contrast to MBConv, Fuse MBConv combines MBConv's 1x1 convolutional layer and 3x3 depthwise convolutional layer into a single step with a smaller expansion ratio, allowing for a lower amount of memory used in the computation, greatly reducing the computational complexity.

Architecture of Gold-YOLO's blocks

The neck structure from Gold-YOLO consists of low semantic fusion and high semantic fusion. The structure of low semantic fusion is shown in Fig. 2, which consists of four sections: Low-FAM, Low-IFM, Low-LAF, and Inject. Firstly, Low-FAM, scales the C2, C3, C4, and C5 with either an average pooling layer or an upsampling (Billinear) method to align to a uniform resolution; Low-IFM limits the input to the embed dimension via Conv (convolutional layer + BN + SiLU activation function) to get the features, the features are then passed through k RepVGG blocks, and next the Conv outputs the features which are finally split into gobal features with the number of channels c3 and

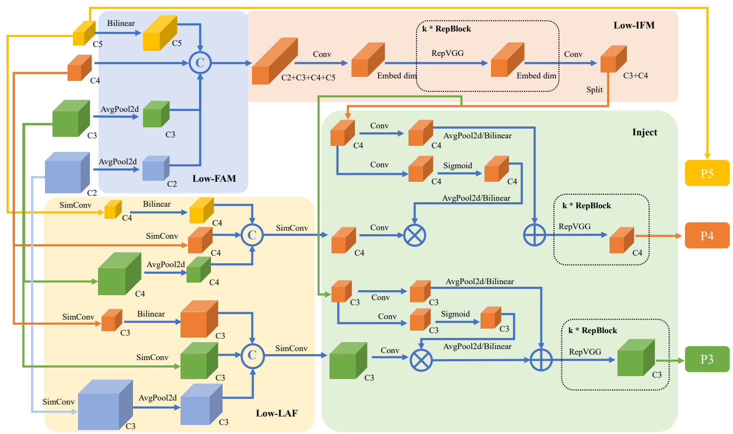


Figure 2: The architecture of low semantic fusion in the neck structure from Gold-YOLO.

c4 respectively; Low-LAF computes C3, C4, C5 and C2, C3, C4 respectively, taking the former as an example, it limits C3, C4, C5 to a uniform channel of c4 by SimConv (convolutional layer + BN + ReLU activation function), and then concatenates C3, C5 through average pooling layer or up-sampling to generate local features with a uniform resolution; In Inject, the local features are passed through SimConv and Conv, multiplied and added to the two branches of the gobal features separately, finally enter into the RepVGG blocks to output P4 feature map. As for C2, C3, and C4, the same procedure is followed to output the P3 feature map; the P5 feature map is directly obtained from C5.

The structure of high semantic fusion is presented in Fig. 3, which is likewise divided into four steps: High-FAM, High-IFM, High-LAF, and Inject. High-FAM unifies P3, P4, and P5 by SimConv, and then uses the same approach in low semantic fusion to unify the resolution and concatenate them; High-IFM employs k transformer blocks, and splits them into global features with channels p4 and p5 respectively; High-LAF and Inject have a similar architecture with the above low semantic fusion's. N3 is directly derived from P3, then N3, N4, and N5 are the final outputs of the neck, which will be input to the detect head for prediction.

Inference results for various deployment tools

We use different deployment tools for inference, and the corresponding inference times of S^3Det are shown in Table 1. It can be observed that the model deployed with TensorRT is the fastest, reaching an average of 29ms/per image speed at INT32 accuracy, almost 3 times as fast as the original.

Inference results for various deployment tools are exhibited in the following Fig. 4. It is shown that the prediction results of the transformed model without reducing accuracy are similar, but using TensorRT at INT8 accuracy loses

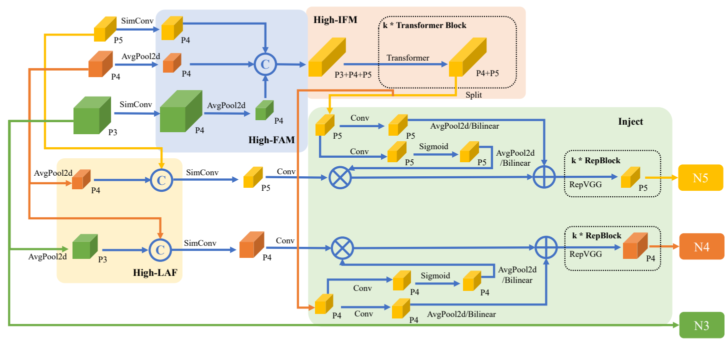


Figure 3: The architecture of high semantic fusion in the neck structure from Gold-YOLO.

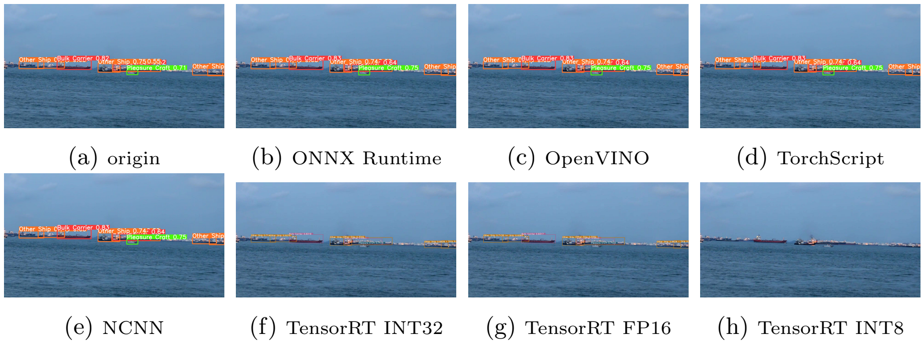
Table 1: Efficiency comparison of various deployment tools.

Deployment Tool	Latency(ms)		
	min	max	mean
origin	99.7	111.6	103.7
ONNX Runtime	934.9	1032.5	964.9
OpenVINO	201.3	243.2	208.2
TorchScript	39.9	44.9	42.9
NCNN	426.7	473.1	465.7
TensorRT INT32	27.8	30.5	29
TensorRT FP16	11.8	13.2	11.9
TensorRT INT8	7.6	17.1	8.8

almost all the predictions. In contrast, using TensorRT at FP16 accuracy can not only promote the inference speed and compress the model but also the loss of accuracy is limited to a certain extent.

Performance of the S^3Det under four extreme weathers

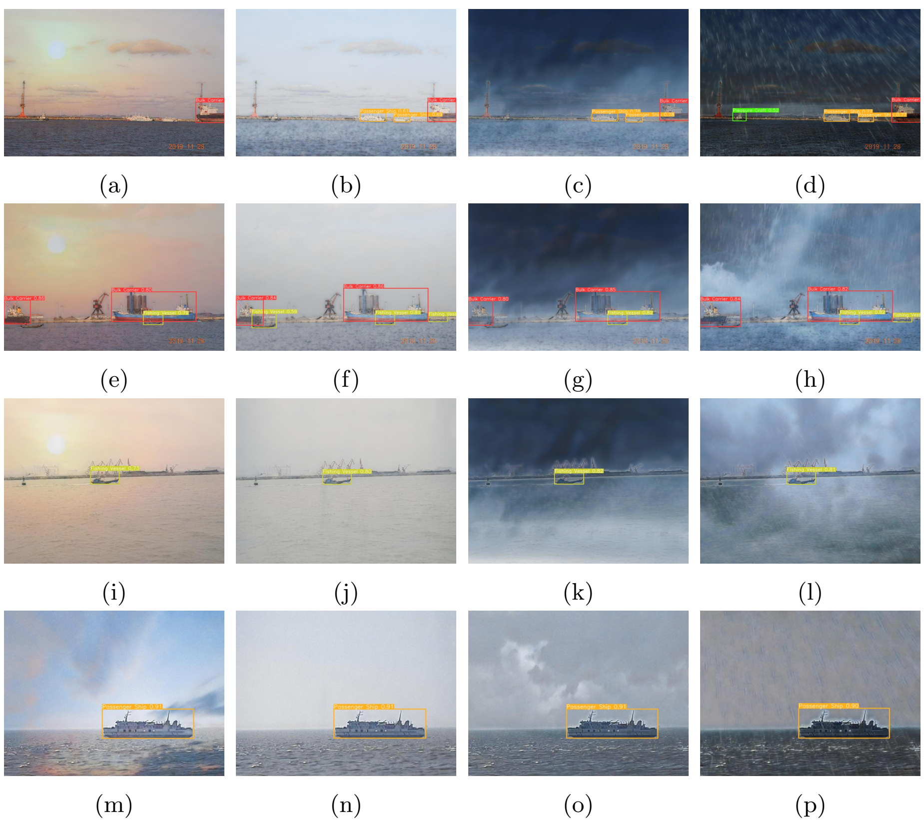
The detection task remains challenging under these weather conditions, with the model experiencing missed detections, particularly in dusk weather where it struggles to detect small-sized ships. Conversely, the model performs relatively better in rainy weather conditions. Additionally, compared to the harbor environment with multiple objects and various obstacles, the model demonstrates superior performance in detecting ships in the open sea.



(a) origin (b) ONNX Runtime (c) OpenVINO (d) TorchScript

(e) NCNN (f) TensorRT INT32 (g) TensorRT FP16 (h) TensorRT INT8

Figure 4: Inference result accelerated up by various deployment tools.



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

Figure 5: S^3Det results at dusk(a,e,i,m), foggy(b,f,j,n), cloudy(c,g,k,o), and rainy(d,h,l,p) days, where (a,b,c,d,e,f,g,h) are harbor collections and (i,j,k,l,m,n,o,p) are open sea collections.