Fourier Angle Alignment for Oriented Object Detection in Remote Sensing

Supplementary Material

This supplementary material provides more details and results that are not included in the main paper due to space limitations. The contents are organized as follows:

- Section S1 exhibits the detailed proof of fourier rotation equivariance, including the steps left out;
- Section S2 gives comparison results on more backbones;
- Section S3 shows ablation experiments on parameter settings for our two modules, FAAFusion and FAA Head.
- Section S4 shows visualization results.

S1. Detailed Mathematical Proof

Suppose $\mathbf{I}(\mathbf{x})$ denotes a 2D signal in the spatial domain. Its Fourier Transform is defined as:

$$F(\boldsymbol{\omega}) = \mathcal{F}\{\mathbf{I}(\mathbf{x})\} = \int_{\mathbb{R}^2} \mathbf{I}(\mathbf{x})\, e^{-j2\pi\boldsymbol{\omega}^\intercal \mathbf{x}}\, d\mathbf{x}, \tag{1}$$

where $\boldsymbol{\omega} = (u, v)^\intercal \in \mathbb{R}^2$ denotes the frequency-domain coordinate. Next, consider rotating $\mathbf{I}$ counterclockwise by an angle ϕ around the origin. The rotated image is given by

$$\mathbf{I}_\phi(\mathbf{x}) = \mathbf{I}(\mathbf{R}_{-\phi}\mathbf{x}), \tag{2}$$

where $\mathbf{R}_\phi$ is the standard 2D rotation matrix:

$$\mathbf{R}_\phi = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}. \tag{3}$$

Note that $\mathbf{R}_{-\phi} = \mathbf{R}_\phi^\intercal$, since rotation matrices are orthogonal. Let $F_\phi(\boldsymbol{\omega})$ denotes the Fourier Transform of the rotated signal $\mathbf{I}_\phi$. By definition,

$$F_\phi(\boldsymbol{\omega}) = \mathcal{F}\{\mathbf{I}_\phi(\mathbf{x})\} = \int_{\mathbb{R}^2} \mathbf{I}(\mathbf{R}_{-\phi}\mathbf{x})\, e^{-j2\pi\boldsymbol{\omega}^\intercal \mathbf{x}}\, d\mathbf{x}. \tag{4}$$

Now we perform a change of variables to simplify the integral. Define a new variable

$$\mathbf{u} = \mathbf{R}_{-\phi}\mathbf{x}. \tag{5}$$

Since $\mathbf{R}_{-\phi}$ is invertible (in fact, orthogonal), we can solve for $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{R}_\phi \mathbf{u}, \tag{6}$$

because $\mathbf{R}_\phi = (\mathbf{R}_{-\phi})^{-1} = \mathbf{R}_{-\phi}^\intercal$.

The Jacobian matrix of the transformation $\mathbf{x} \to \mathbf{u}$ is $\mathbf{R}_{-\phi}$, and its determinant is

$$\det(\mathbf{R}_{-\phi}) = \cos^2\phi + \sin^2\phi = 1. \tag{7}$$

Therefore, the differential element is invariant under this rotation:

$$d\mathbf{x} = |\det(\mathbf{R}_\phi)|\, d\mathbf{u} = d\mathbf{u}. \tag{8}$$

Substituting $\mathbf{x} = \mathbf{R}_\phi\mathbf{u}$ and $d\mathbf{x} = d\mathbf{u}$ into the expression for $F_\phi(\boldsymbol{\omega})$, we obtain

$$F_\phi(\boldsymbol{\omega}) = \int_{\mathbb{R}^2} \mathbf{I}(\mathbf{u})\, e^{-j2\pi\boldsymbol{\omega}^\intercal(\mathbf{R}_\phi\mathbf{u})}\, d\mathbf{u}. \tag{9}$$

Now consider the exponent term $\boldsymbol{\omega}^\intercal(\mathbf{R}_\phi\mathbf{u})$. Using the fundamental identity for transposes of matrix-vector products,

$$(\mathbf{A}\mathbf{b})^\intercal = \mathbf{b}^\intercal\mathbf{A}^\intercal, \tag{10}$$

we can rewrite the scalar product as follows:

$$\begin{aligned} \boldsymbol{\omega}^\intercal(\mathbf{R}_\phi\mathbf{u}) &= (\boldsymbol{\omega}^\intercal\mathbf{R}_\phi)\mathbf{u} \\ &= (\mathbf{R}_\phi^\intercal\boldsymbol{\omega})^\intercal\mathbf{u}. \end{aligned} \tag{11}$$

Thus, the exponential becomes

$$e^{-j2\pi\boldsymbol{\omega}^\intercal(\mathbf{R}_\phi\mathbf{u})} = e^{-j2\pi(\mathbf{R}_\phi^\intercal\boldsymbol{\omega})^\intercal\mathbf{u}}. \tag{12}$$

Plugging this back into the integral yields

$$F_\phi(\boldsymbol{\omega}) = \int_{\mathbb{R}^2} \mathbf{I}(\mathbf{u})\, e^{-j2\pi(\mathbf{R}_\phi^\intercal\boldsymbol{\omega})^\intercal\mathbf{u}}\, d\mathbf{u}. \tag{13}$$

But the right-hand side is precisely the definition of the Fourier Transform of $\mathbf{I}$ evaluated at the frequency $\mathbf{R}_\phi^\intercal\boldsymbol{\omega}$. Therefore,

$$F_\phi(\boldsymbol{\omega}) = F(\mathbf{R}_\phi^\intercal\boldsymbol{\omega}). \tag{14}$$

Finally, using the orthogonality of rotation matrices ($\mathbf{R}_\phi^\intercal = \mathbf{R}_{-\phi}$), we conclude:

$$F_\phi(\boldsymbol{\omega}) = F(\mathbf{R}_{-\phi}\boldsymbol{\omega}). \tag{15}$$

This result demonstrates that rotating the 2D spatial-domain signal by angle ϕ counterclockwise induces an identical counterclockwise rotation of its Fourier spectrum. Hence, the 2D Fourier Transform is *equivariant* with respect to planar rotations.

S2. More Experiments on Backbone

In the main paper, we used three representative backbones to show that our method works well. To prove that our method is widely applicable and easy to use with different models, we test it on more backbones in this section. We use ResNet18 [3], ResNet34 [3], ResNet101 [3], and Swin-Transformer [5]. We train and test on DOTA-v1.0 [2] using the same implementation settings as the main paper.

As shown in Table S1, our method improves performance on all these four backbones. Among these methods using Swin-Transformer [5] as the backbone, our method

Table S1. Results with more backbones on DOTA-v1.0 [2] dataset with single scale training and testing.

Method	Backbone	PL	BD	BR	GTF	SV	LV	SH	TC	BC	ST	SBF	RA	HA	SP	HC	mAP
O-RCNN [6]	ResNet18	89.36	80.89	51.27	72.47	77.77	81.22	87.58	90.90	87.47	83.53	58.13	63.09	67.91	69.79	50.27	74.11
O-RCNN [6] + ours	ResNet18	89.42	81.04	51.14	71.83	77.75	81.87	87.65	90.90	87.29	84.54	59.73	59.84	72.37	68.88	56.00	**74.68**
O-RCNN [6]	ResNet34	89.16	83.71	52.76	70.85	78.19	81.81	87.68	90.90	86.80	84.94	63.64	62.35	73.74	70.37	49.55	75.10
O-RCNN [6] + ours	ResNet34	89.16	82.84	53.15	76.17	77.34	81.86	87.87	90.90	87.42	84.80	62.86	63.96	73.61	69.47	54.09	**75.70**
O-RCNN [6]	ResNet101	89.13	81.86	54.75	73.72	78.26	78.24	87.55	90.89	86.01	84.45	64.62	66.51	75.58	68.23	49.65	75.30
O-RCNN [6] + ours	ResNet101	88.95	85.06	53.84	72.04	78.47	77.85	87.99	90.90	86.00	84.79	63.60	68.00	75.88	69.62	52.31	**75.69**
O-RCNN [6]	Swin-T	89.22	83.26	54.15	75.52	78.94	84.59	88.14	90.90	87.46	86.38	66.09	65.73	75.13	71.05	64.04	77.37
O-RCNN [6] + ours	Swin-T	89.45	84.24	54.12	76.81	79.28	84.30	88.19	90.90	87.86	86.20	67.69	66.91	76.04	71.70	63.74	**77.83**

outperforms the previous state-of-tha-art method RoI Transformer [1] by 0.32% and also outperforms the latest method ReDiffDet [7] by 1.21%. Our method works well on both CNN-based and Transformer-based backbones, which shows that our method is widely applicable.

S3. Futher Analysis

In the main paper, we used fixed parameter settings without explaining the reason. In this section, we test different settings for FAAFusion and FAA Head, and analyze how these settings affect performance. To test the effectiveness more conveniently, we use the `train` set of the DOTA-v1.0[2] dataset for training and the `val` set for testing.

S3.1. Settings of FAAFusion

Our FAAFusion is a module used in the Feature Pyramid Network (FPN) [4] to combine low-level and high-level features. First, FAAFusion takes two inputs: low-level features and upsampled high-level features. Both have 256 channels. To reduce computation, we use a 1×1 convolution to reduce the channel number to middle dimension. Then, we use unfold to extract local patches with a sliding window. Each patch pair contains one low-level patch and one high-level patch, matched one-to-one. We apply FFT to each patch pair and estimate the main direction for both low-level and high-level patches. Then we rotate the high-level patch to match the direction of the low-level patch. After alignment, we use fold to put all aligned high-level patches back into a full feature map, with a 1×1 convolution to increase the channel dimension to 256. We add the original (unaligned) high-level features as a residual connection to the aligned ones. Then we add this final high-level feature element-wise to the low-level feature to get the fused output.

In this process, three points are worth discussing: the value of middle dimension, the size of the sliding window in unfold and where in the FPN [4] we apply FAAFusion.

About the middle dimension. For the intermediate dimension, we want to reduce computation while keeping accuracy as high as possible. We test four values: 32, 64, 90, and 128. We report mAP value, along with the number of parameters and FLOPs. As shown in Table S2, compared to the baseline, when the middle dimension is 32, the improvement is small. When the middle dimension is 64, 90, or 128, the results are similar and all show clear improvement over the baseline. Considering computation cost, we choose 64 as the middle dimension in the main paper.

Table S2. Comparisons among different middle dimension of our FAAFusion. Baseline means the original result without FAAFusion. The FAAFusion is applied within the Oriented R-CNN [6] framework. All the models are retrained and tested on a single RTX 3090. The Params and FLOPs are calculated with MMRotate [8] toolbox.

Mid-dim	Backbone	#Params↓	#FLOPs↓	mAP↑
Baseline	ResNet50[3]	41.14M	211.43G	71.7
32	ResNet50[3]	45.87M	213.44G	71.8
64	ResNet50[3]	45.88M	214.25G	72.7
90	ResNet50[3]	45.93M	217.17G	72.5
128	ResNet50[3]	45.96M	219.08G	72.6

Table S3. Comparisons among different kernel size of our FAAFusion unfolding window. The FAAFusion is applied within the Oriented R-CNN [6] framework. All the models are retrained and tested on a single RTX 3090. The Params and FLOPs are calculated with MMRotate [8] toolbox.

Kernel size	Backbone	#Params↓	#FLOPs↓	mAP↑
3×3	ResNet50[3]	45.88M	214.22G	71.8
5×5	ResNet50[3]	45.88M	214.24G	71.9
7×7	ResNet50[3]	45.88M	214.25G	72.7
9×9	ResNet50[3]	45.88M	214.27G	72.5

About the kernel size of the unfolding window. We test kernel sizes of 3, 5, 7, and 9. We record the number of parameters, FLOPs, and mAP for each. As shown in Table S3,

Table S4. Comparisons among different locations of our FAAFusion. ✔ denotes replacing element-wise add with FAAFusion. The FAAFusion is applied within the Oriented R-CNN [6] framework. All the models are retrained and tested on a single RTX 3090. The Params and FLOPs are calculated with MMRotate [8] toolbox.

P3 → P2	P4 → P3	P5 → P4	#Params↓	#FLOPs↓	mAP↑
✘	✘	✘	41.14M	211.43G	71.7
✔	✘	✘	45.88M	214.25G	72.7
✔	✔	✘	45.91M	214.65G	72.5
✔	✔	✔	45.93M	214.75G	72.7

when the kernel size is 3 or 5, the window captures limited texture and direction information, so the results are close to the baseline. When the kernel size is 7 or 9, the window captures more useful information, so both give clear improvements. Our method uses sliding windows for FFT and alignment, and has no extra parameters. So changing the kernel size does not change the number of parameters. It only slightly changes the computation cost. But since most objects in detection are small, a large window may bring in noise. This may not hurt much in spatial domain, but in frequency domain, it can interfere with angle estimation. So we choose a smaller window. Therefore, in the main paper, we set kernel size to 7.

About the location of FAAFusion. FAAFusion is designed to use low-level features to correct angle information in high-level features, so we replace element-wise add with FAAFusion in a bottom-up way. We use the original FPN [4] as the baseline, and replace from the lowest fusion to the highest. As shown in Table S4, when we replace element-wise add with FAAFusion, all three kinds of replacement improve performance, and the results are similar. This is because the layer with the most angle information is $C2$. Using $C2$ to guide the fusion of $P3$ to produce $P2$ gives the highest improvement in angle information. $C3$ and $C4$ are higher-level features that contain less angle information, especially for small objects. As indicated from the results, applying FAAFusion at higher levels brings little improvement but increases both parameters and computation. Therefore, in the main paper, we only use FAAFusion when fusing $C2$ and $P3$ to produce $P2$.

S3.2. Settings of FAA Head

Our FAA Head is a new detection head applied directly to RoI-based [1] detectors. First, FAA Head takes RoI features as input. Then it applies FFT to get the spectrum. It estimates the main direction of energy distribution. Next, it rotates the features to align this main direction to a fixed angle, making the features somewhat rotation-invariant. But the original features (before alignment) still contain useful direction and position information that helps with bounding box regression. So we feed both the original and aligned features into the following fully connected layers for classification and regression.

In this process, two points worth discussing: how we rotate the features and how we combine the original and aligned features.

Table S5. Results of different settings of our FAA Head. The FAA Head is applied within the Oriented R-CNN [6] framework. All the models are retrained and tested on a single RTX 3090. The Params and FLOPs are calculated with MMRotate [8] toolbox.

Method	Backbone	#Params↓	#FLOPs↓	mAP↑
Rotation domain				
Frequent	ResNet50[3]	58.51M	217.17G	72.3
Spatial	ResNet50[3]	58.51M	214.90G	73.1
Residual setting				
Concat	ResNet50[3]	74.57M	230.96G	72.6
Add	ResNet50[3]	58.51M	214.90G	73.1

About the rotation domain. We test two rotation methods: (1) Rotate the spectrum in the frequency domain, then apply inverse FFT to get back to spatial domain. (2) Rotate the spatial feature map directly. We report and compare parameters, FLOPs, and mAP for both. In theory, both methods give the same result. But since images are discrete signals, rotation is done by interpolation. Rotating in frequency domain causes signal loss, and more loss happens during inverse FFT. As is shown in Table S5 rotating in spatial domain works better. Also, spatial rotation skips the inverse FFT, so it is faster.

About the residual setting. For the combination of the original and aligned features, we test two methods: concatenate them along the channel dimension and add the original features as a residual (element-wise addition). We conduct the experiment and report parameters, FLOPs, and mAP for both. As shown in Table S5, adding as residual gives better results and uses fewer parameters and FLOPs. This is because concatenation doubles the input dimension after flattening, which greatly increases the number of parameters and computation in the fully connected layers. Therefore, in the main paper, we use residual addition as the combination method.

S4. Visualizations

In this section, we show some comparison results of object detection. We use Oriented R-CNN [6] with ResNet50

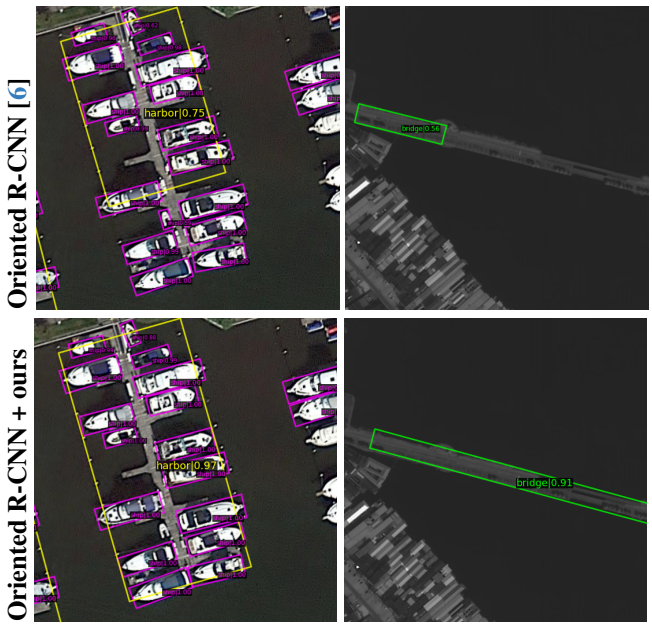

Figure S1. Detection results comparison. Our method detects objects more completely.

[3] as the backbone and compare the detection results before and after adding our FAAFusion and FAA Head. Our method performs better in four cases:

- As shown in Figure S1, our method detects objects more completely.
- As shown in Figure S2, our method gives higher classification scores for detected objects.
- As shown in Figure S3, our method has fewer false detections.
- As shown in Figure S4, our method has fewer missed detections.

References

[1] Jian Ding, Nan Xue, Yang Long, Gui-Song Xia, and Qikai Lu. Learning roi transformer for oriented object detection in aerial images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2849–2858, 2019. 2, 3

[2] Jian Ding, Nan Xue, Gui-Song Xia, Xiang Bai, Wen Yang, Michael Ying Yang, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Object detection in aerial images: A large-scale benchmark and challenges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7778–7796, 2022. 1, 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1, 2, 3, 4

[4] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 936–944, 2017. 2, 3

[5] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *IEEE International Conference on Computer Vision*, pages 10012–10022, 2021. 1

[6] Xingxing Xie, Gong Cheng, Jiabao Wang, Xiwen Yao, Ke Li, and Junwei Han. Oriented R-CNN and beyond. *International Journal of Computer Vision*, 132:2420–2442, 2024. 2, 3, 4, 5, 6

[7] Jiaqi Zhao, Zeyu Ding, Yong Zhou, Hancheng Zhu, Wen-Liang Du, and Rui Yao. Rediffdet: Rotation-equivariant diffusion model for oriented object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 24429–24439, 2025. 2

[8] Yue Zhou, Xue Yang, Gefan Zhang, Jiabao Wang, Yanyi Liu, Liping Hou, Xue Jiang, Xingzhao Liu, Junchi Yan, Chengqi Lyu, Wenwei Zhang, and Kai Chen. Mmrotate: A rotated object detection benchmark using pytorch. In *ACM International Conference on Multimedia*, pages 7331–7334, 2022. 2, 3

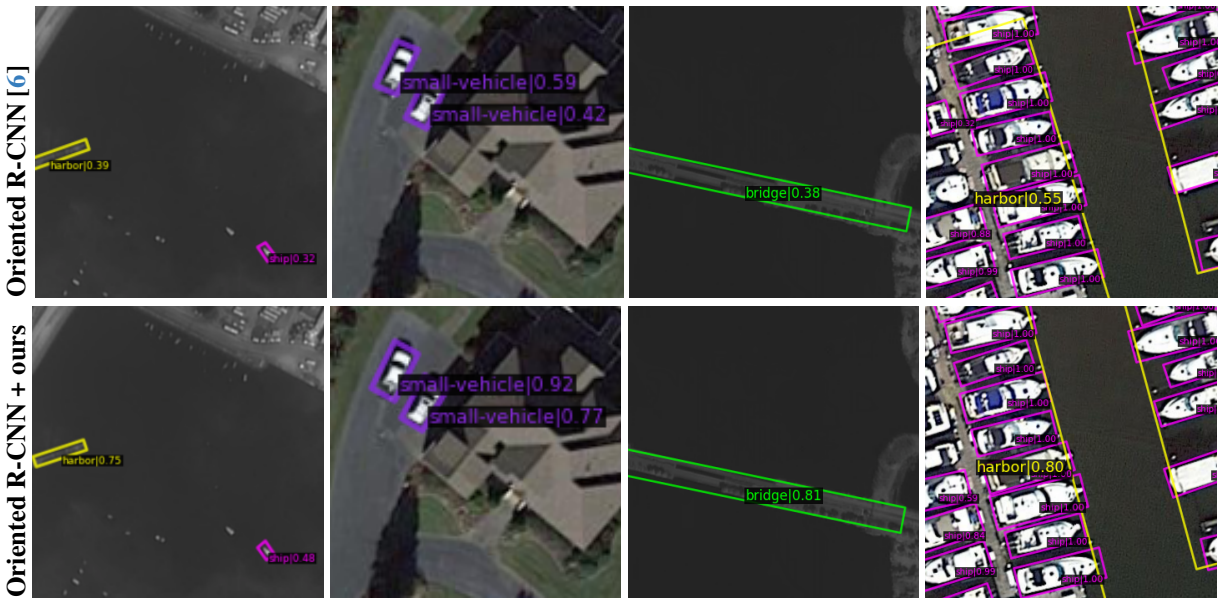

Figure S2. Detection results comparison. Our methods have higher classification score.

Figure S3. Detection results comparison. Our methods have less wrong detection.

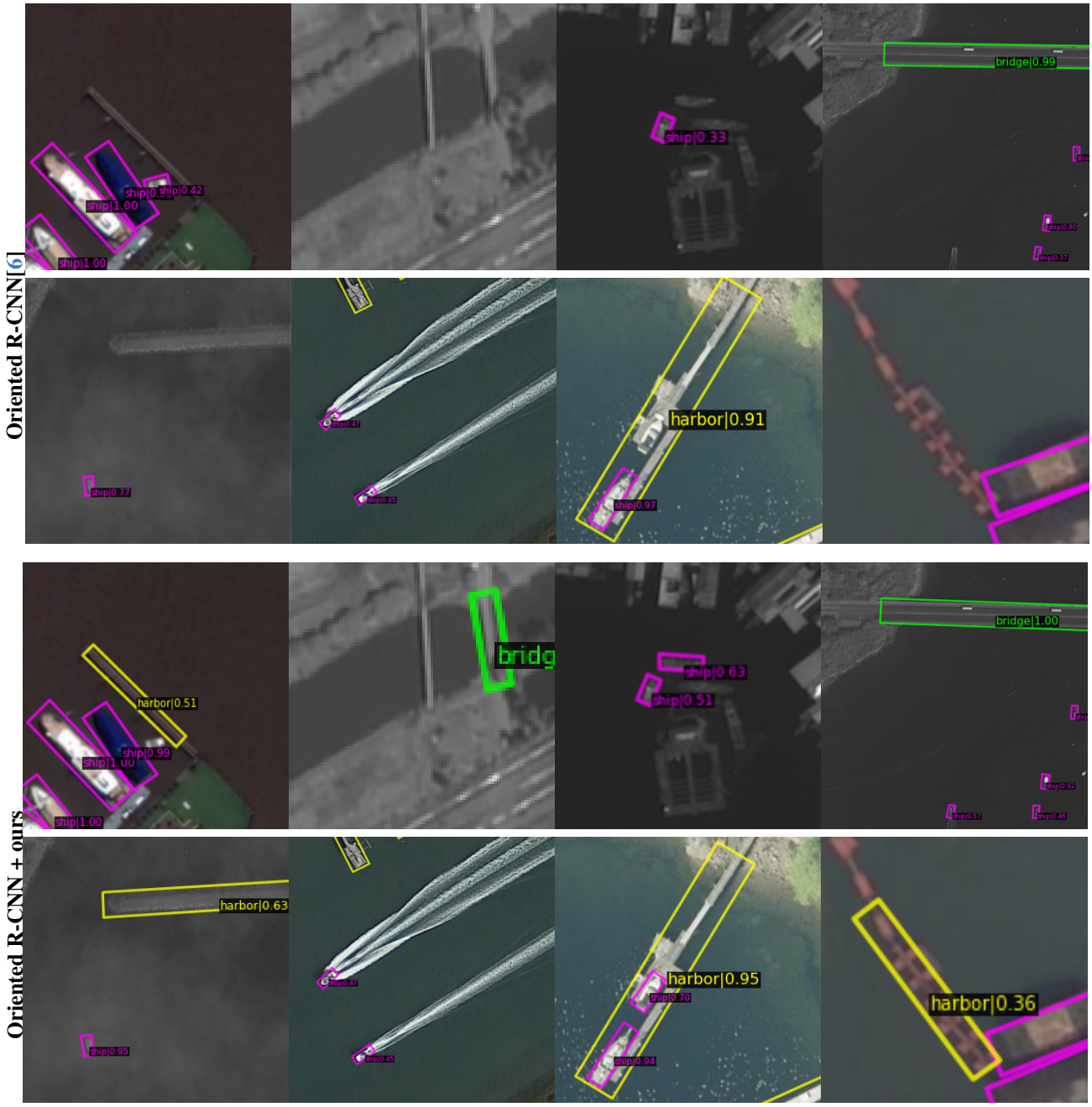

Figure S4. Detection results comparison. Our method has fewer missed detections.