Rotated Object Detection in Aerial Images

KONSTANTINOS PAPADAKIS
K.I.PAPADAKIS@GMAIL.COM

The DOTA v1.0 Dataset

The DOTA v1.0 Dataset

A large-scale dataset for rotated object detection in aerial images

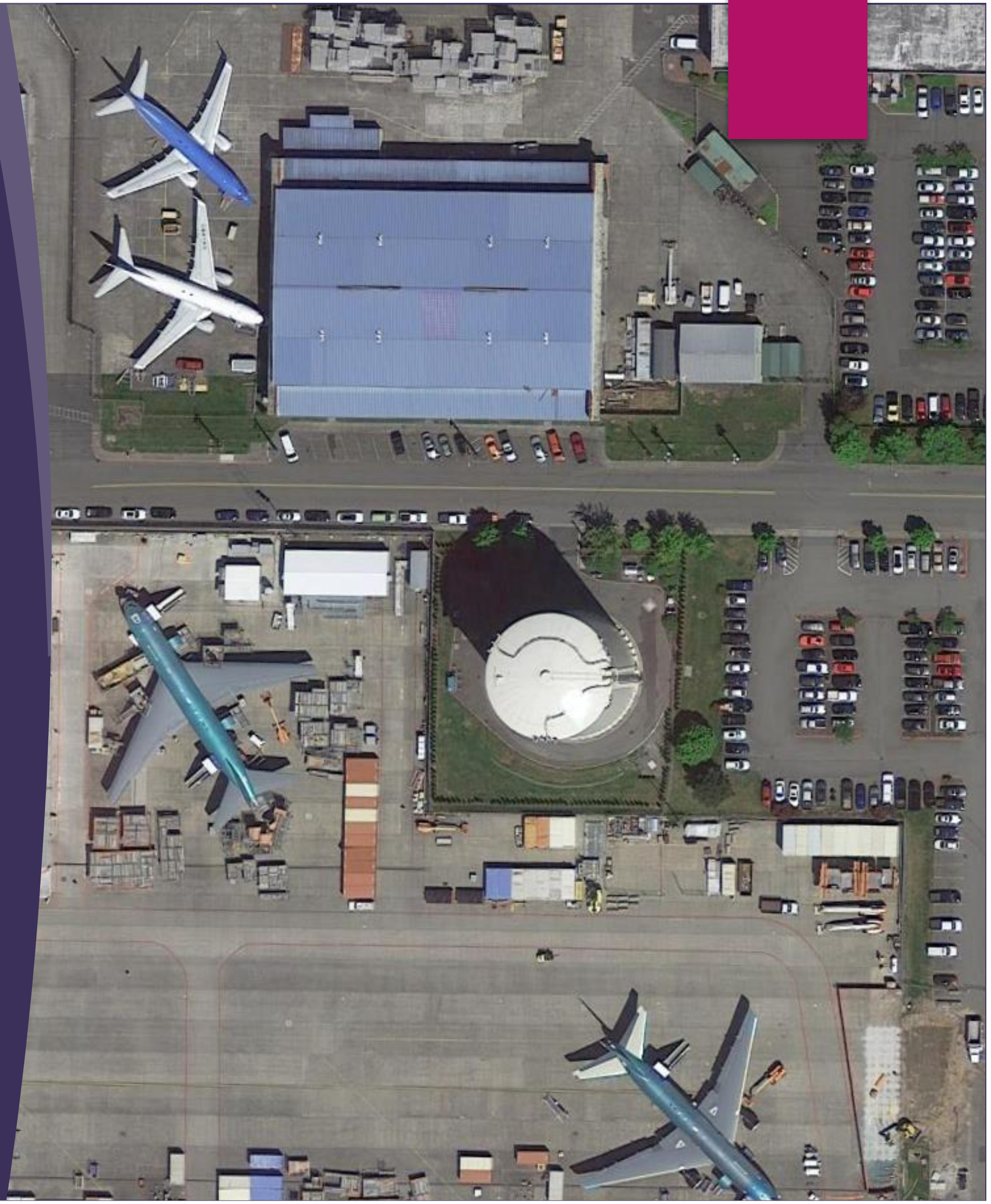
Commonly used as a benchmark

Image size ranging from 800×800 to 20,000×20,000 pixels

20GB of images

2,806 images, 188,282 instances

1/2, 1/6, 1/3 train-validation-test split



Oriented vs Horizontal Bounding Boxes

- In aerial images, objects are often arbitrarily oriented due to the bird's eye view and larger scale variations when compared to natural images.
- Crowded instances represented by horizontal bounding boxes (HBBs) are difficult to distinguish.
- Oriented Bounding Boxes (OBBs) are a better fit for aerial images.


Large-vehicle
Swimming pool
Helicopter
Bridge
Plane
Ship
Soccer ball field
Basketball court
Airport
Container crane
Ground track field
Small vehicle
Harbor
Baseball diamond
Tennis court
Roundabout
Storage tank
Helipad

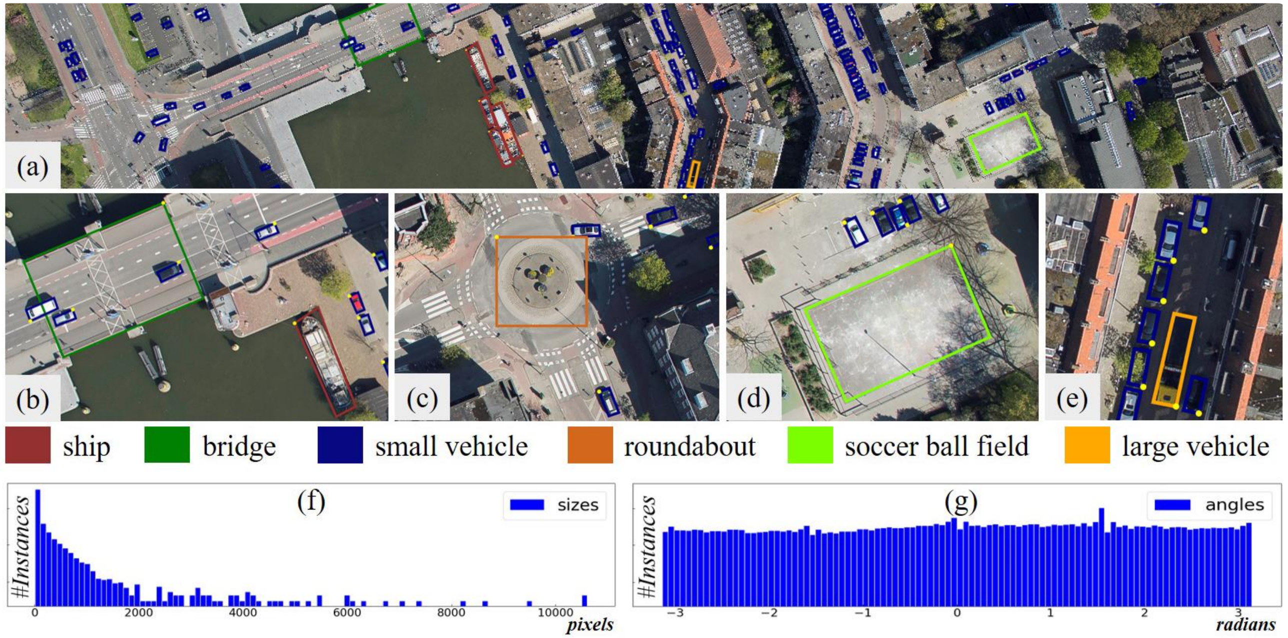
15 classes

188,282 Instances

Category	Number of Instances
Plane	14,085
Baseball Diamond	1,130
Bridge	3,760
Ground Track Field	678
Small Vehicle	48,891
Large Vehicle	31,613
Ship	52,516
Tennis Court	4,654
Baseball Court	954
Storage Tank	11,794
Soccer Ball Field	720
Roundabout	871
Harbor	12,287
Swimming Pool	3,507
Helicopter	822

Image Properties

- (a) A typical image in DOTA consisting of many instances from multiple categories.
- (b), (c), (d), (e) are cropped from the source image.
- (f) and (g) exhibit the size and orientation histograms, respectively, for all instances.
- Source: https://arxiv.org/abs/2102.12219



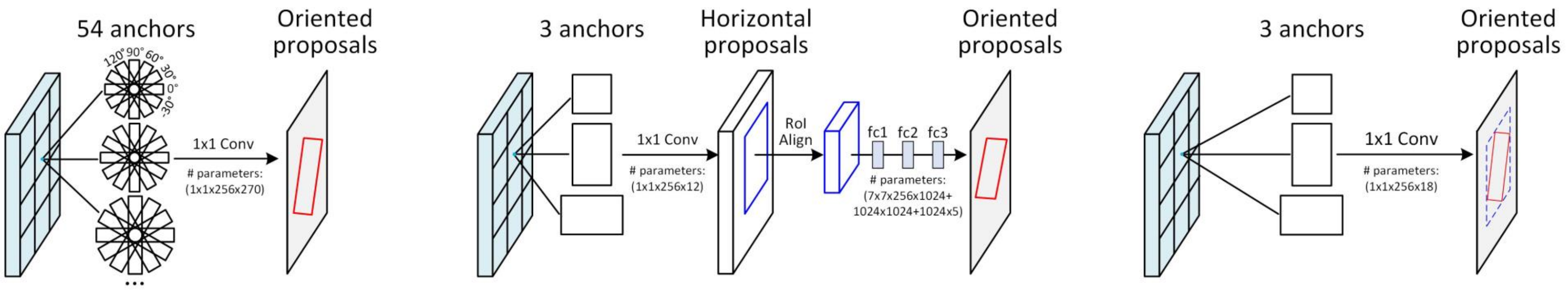
Oriented Object Detection Algorithms

Benchmarks

- Commonly used models are available in the MMRotate toolbox https://mmrotate.readthedocs.io/en/latest/model_zoo.html
- Benchmarks of popular models using ResNet50 as their backbone and 1024×1024 patches with 200 overlap are shown in the table (source: MMRotate)

Model	mAP	Inference FPS	Year
Rotated RetinaNet	68.42	16.9	2017
Rotated ATSS	70.64	18.2	2020
Gliding Vertex	73.23	16.4	2020
Rotated Faster RCNN	73.40	16.5	2017
Oriented RCNN	75.69	16.2	2021
RoI Transformer	76.08	14.4	2019

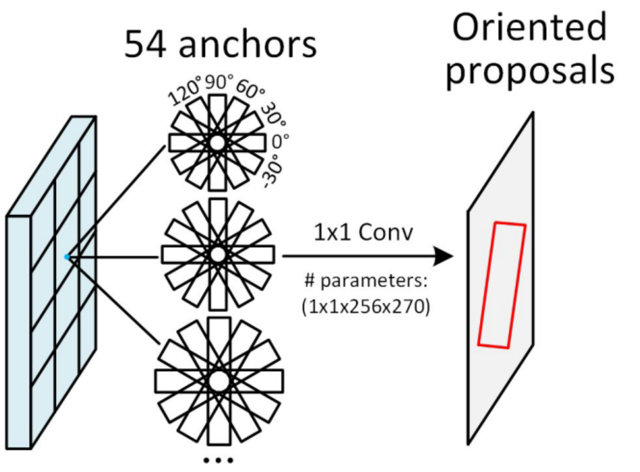
RPN variations



(a) Rotated RPN (b) RoI Transformer+ (c) Oriented RPN

Rotated RetinaNet and Rotated Faster RCNN

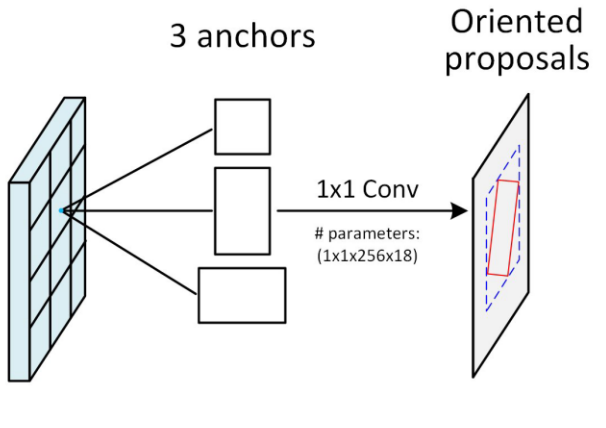
- Created by modifying the original HBB architectures
- The RPN, instead of generating a fixed set of horizontal anchors, it now generates a fixed set of oriented anchors. These anchors are described by their center, height, width and angle.
- The bounding box regressors now also include an angle delta.
- The RoI Alignment is modified to Rotated RoI Alignment and now works with OBBs.



(a) Rotated RPN

Oriented RCNN

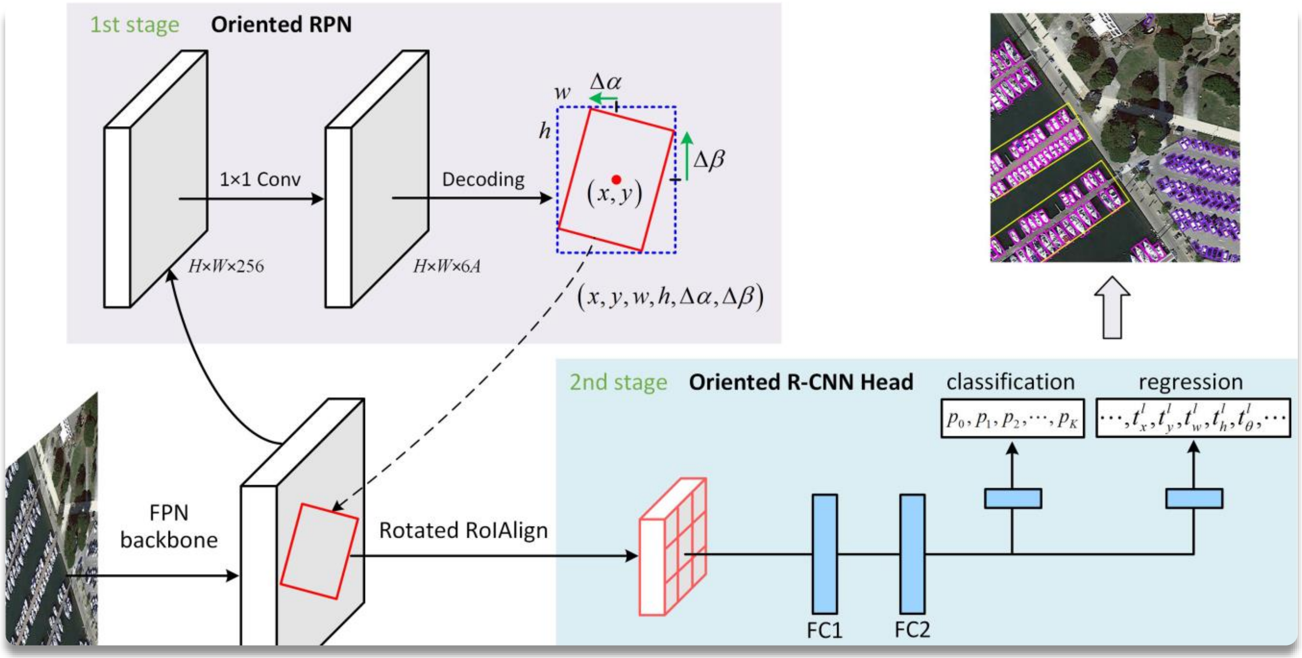
- Like Rotated Faster RCNN but with different RPN.
- The anchors are horizontal, but the proposals are parallelograms contained inside the anchors.



(c) Oriented RPN

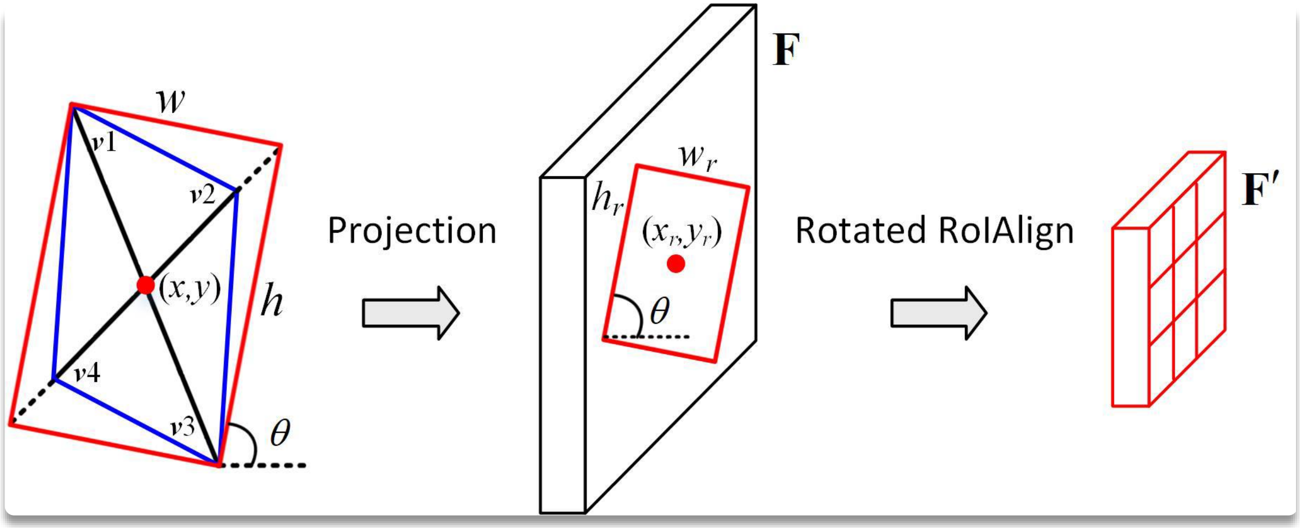
Oriented RCNN

- The box regressor generates a delta for the center, height and width of the horizontal anchor, as well as two deltas that are used to create offsets Δα and Δβ relative to the midpoints of the top and right sides of the horizontal anchor.
- Each proposal is a parallelogram represented by 6 numbers: the center, height, width and two offsets. This representation is called *Midpoint Offset Representation*.



Oriented RCNN

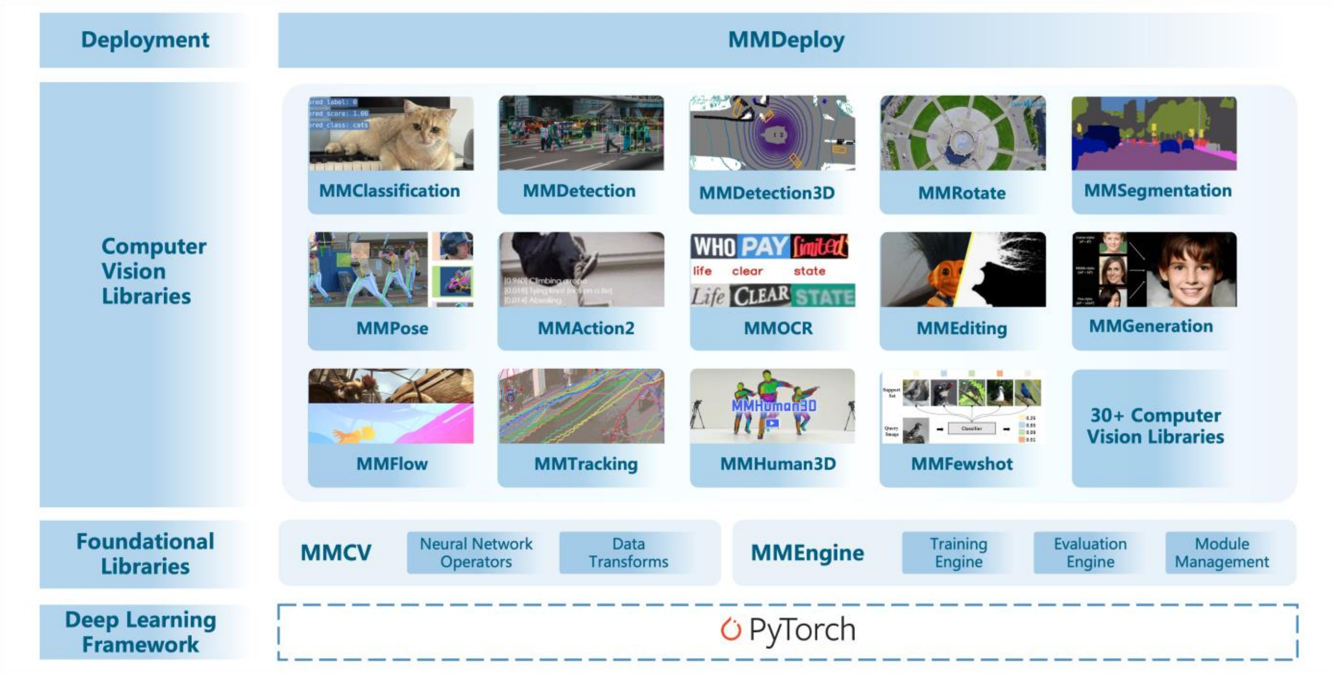
- To utilize Rotated RoI Alignment, the parallelogram proposals are transformed to rotated rectangles by extending the shorter side of the parallelogram.
- The remaining stages of the model are the same as Rotated Faster RCNN.



The MMRotate Library

The MMRotate Library

- MMRotate is part of a bigger project called OpenMMLab, and its application is rotated object detection.
- It is based on MMDetection, which in turn is based on MMCV, a library based on PyTorch.



Configuration Files

- To use any OpenMMLab library, one must specify configuration files (JSON, YAML, or Python files) which contain instructions for the data loading process, the model and the runtime.
- Part of a Python configuration is shown in the image

```
angle_version = 'oc'  # The angle version
model = dict(
    type='RotatedRetinaNet',  # The name of detector
    backbone=dict(  # The config of backbone
        type='ResNet',  # The type of the backbone
        depth=50,  # The depth of backbone
        num_stages=4,  # Number of stages of the backbone.
        out_indices=(0, 1, 2, 3),  # The index of output feature maps produced in each stages
        frozen_stages=1,  # The weights in the first 1 stage are fronzen
        zero_init_residual=False,  # Whether to use zero init for last norm layer in resblocks to let them behave as identity.
        norm_cfg=dict(  # The config of normalization layers.
            type='BN',  # Type of norm layer, usually it is BN or GN
            requires_grad=True),  # Whether to train the gamma and beta in BN
        norm_eval=True,  # Whether to freeze the statistics in BN
        style='pytorch',  # The style of backbone, 'pytorch' means that stride 2 layers are in 3x3 conv, 'caffe' means stride 2 layers are in 1x1 convs.
        init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),  # The ImageNet pretrained backbone to be loaded
    neck=dict(
        type='FPN',  # The neck of detector is FPN. We also support 'ReFPN'
        in_channels=[256, 512, 1024, 2048],  # The input channels, this is consistent with the output channels of backbone
        out_channels=256,  # The output channels of each level of the pyramid feature map
        start_level=1,  # Index of the start input backbone level used to build the feature pyramid
        add_extra_convs='on_input',  # It specifies the source feature map of the extra convs
        num_outs=5),  # The number of output scales
    bbox_head=dict(
        type='RotatedRetinaHead',# The type of bbox head is 'RRetinaHead'
        num_classes=15,  # Number of classes for classification
```

Preprocessing a Dataset

- The dataset needs to be an appropriate format. Most often it is the case that one transforms a dataset into the “DOTA format” and utilize the predefined DOTADataset class.
- If the images are large, one needs to crop the dataset into patches. This can be achieved by using scripts provided in the MMRotate GitHub repository. The script is `tools/data/dota/split/img_split.py`
- MMRotate also provides functions to reassembly patch predictions by using Non-maximum Suppression (NMS).

Training and Testing a Model

- After specifying the configurations and bringing the dataset into an appropriate format, the model can be trained by running a script provided in the MMRotate GitHub repository with the configuration file path as an argument. The script is `tools/train.py`
- The model can be tested by running `tools/test.py` with the configuration and model checkpoint paths.

Results

Data Preprocessing

- Each image was cropped into 1024 × 1024 patches with an overlap of 200.
- Before feeding the image to the network, the following transforms were applied:
  - Transform gray images to RGB by repeating the gray channel
  - Image normalization using the default ResNet50 transforms
  - Resize the image to 1024 ×1024 using bilinear interpolation (training only)
  - Random horizontal, vertical and diagonal flip (training only)

Model Structure

Rotated RetinaNet

- ResNet50 backbone pretrained on ImageNet with the final 4 out 5 layers being trainable.
- The anchors are all with zero angles, due to MMRotate's implementation (see `mmrotate.core.anchor.anchor_generator.AnchorGenerator`)
- Samples are assigned as positive if they have OBBs with IoU > 0.5 with the ground truth OBBs, and negative (background) if IoU < 0.4.
- Focal loss gamma=2.0 and alpha=0.25.

Oriented RCNN

- The same backbone as Rotated RetinaNet.
- RPN IoU thresholds are 0.7 for positives and 0.3 for negatives.
- RCNN Head IoU thresholds are 0.5 for positives and 0.5 for negatives (meaning that no RPN proposal is ignored)
- Oriented RCNN can have stricter thresholds for its RPN since it has the easier tasks of learning only two classes: object and background.

Training

Rotated RetinaNet

- The model was trained for 13 epochs (7 hours and 13 minutes).
- The best mAP was achieved on epoch 12.
- The batch size was equal to 3.
- Simple SGD with momentum 0.9 was used.
- Weight decay with a coefficient of 0.0001.
- Gradient clipping when L2 norm > 35.
- The learning rate starts at 0.025/3 and grows linearly to 0.025 after 500 iterations (*warmup*). On epochs 8 and 11, the learning rate decays by a factor of 0.1.

Oriented RCNN

- The same training configurations as in Rotated RetinaNet was used, except that the learning rate starts at 0.005 instead of 0.0025.
- 13 epochs took 8 hours to train.
- The best mAP was achieved on epoch 11.

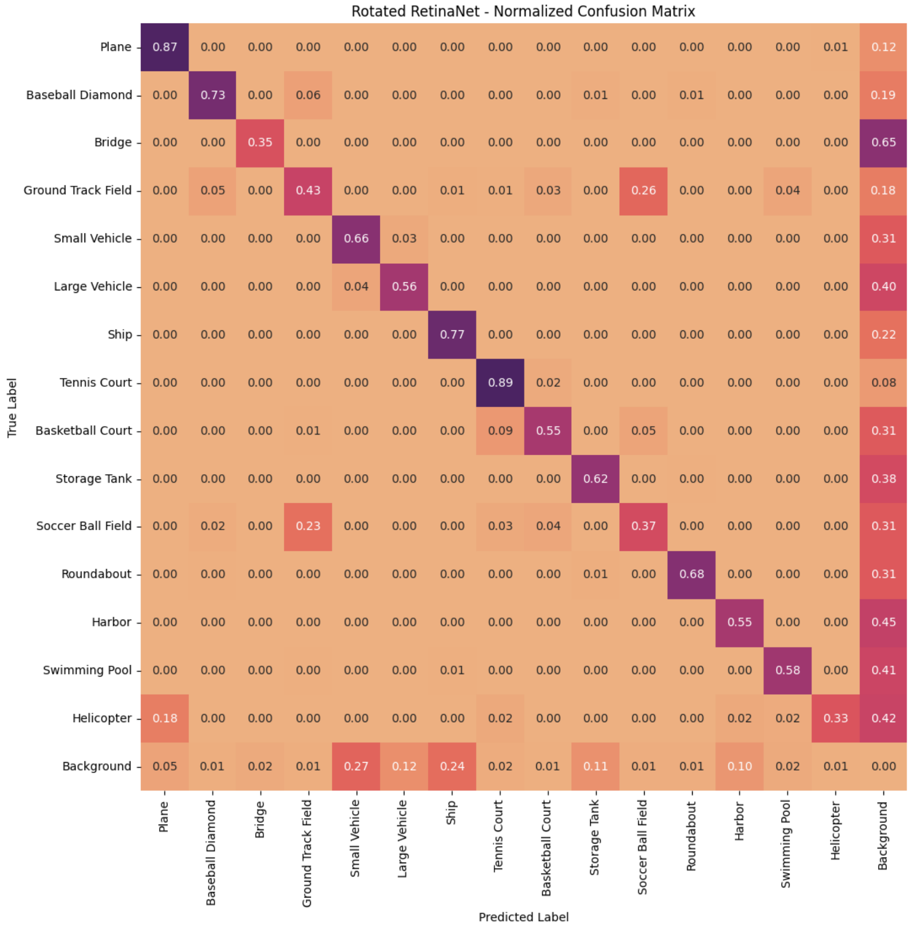
Metrics

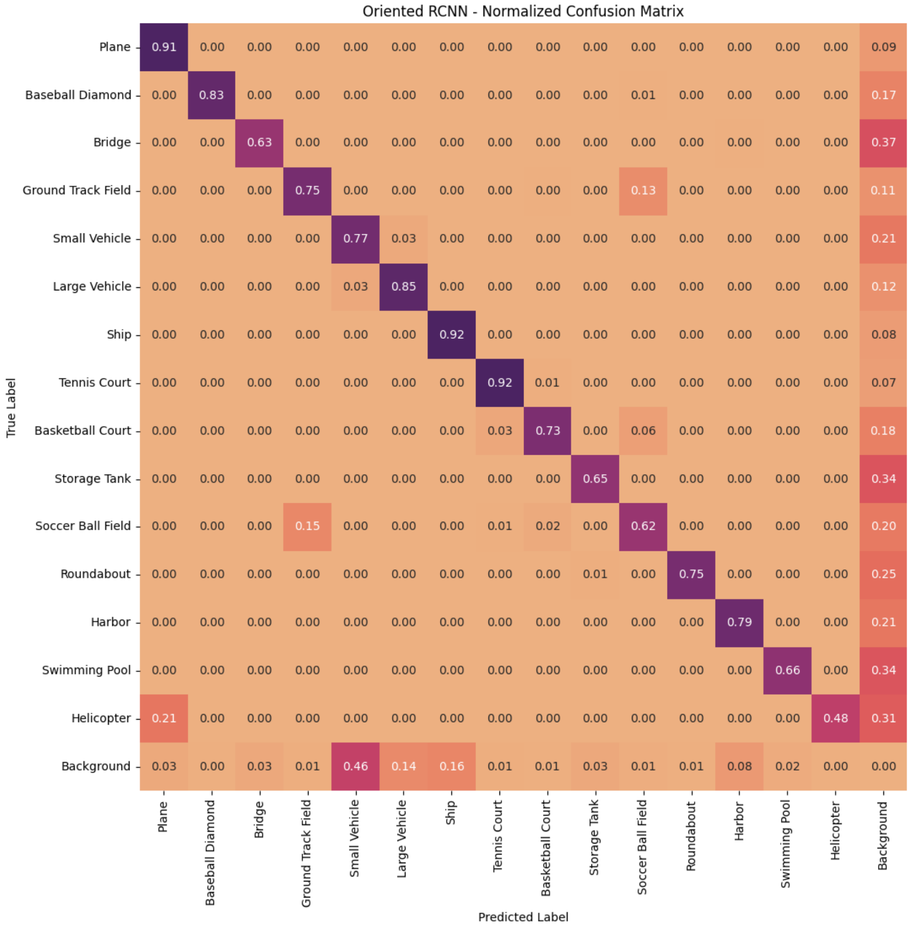
Class	Ground Truths	Detections *Rotated RetinaNet*	Detections *Oriented RCNN*	Recall *Rotated RetinaNet*	Recall *Oriented RCNN*	Average Precision *Rotated RetinaNet*	Average Precision *Oriented RCNN*
Plane	4449	12021	5150	0.911	0.929	0.876	0.893
Baseball Diamond	358	5249	591	0.902	0.866	0.757	0.754
Bridge	783	26870	1945	0.566	0.681	0.341	0.511
Ground Track Field	212	9351	709	0.892	0.901	0.596	0.776
Small Vehicle	10579	114057	27731	0.845	0.843	0.655	0.692
Large Vehicle	8819	71059	16161	0.825	0.924	0.664	0.848
Ship	18537	48742	22256	0.865	0.939	0.777	0.892
Tennis Court	1512	9414	1931	0.947	0.943	0.905	0.908
Basketball Court	266	4469	594	0.793	0.872	0.614	0.742
Storage Tank	4740	23401	4627	0.684	0.688	0.605	0.626
Soccer Ball Field	251	5593	940	0.685	0.833	0.485	0.648
Roundabout	275	6147	600	0.782	0.785	0.631	0.683
Harbor	4167	23615	6600	0.719	0.82	0.585	0.74
Swimming Pool	732	9325	1173	0.697	0.745	0.524	0.579
Helicopter	122	9987	353	0.656	0.705	0.372	0.558
Mean	**3720**	**25287**	**6091**	**0.785**	**0.832**	**0.626**	**0.723**

Metrics

- Rotated RetinaNet has many more detections which is expected since it has smaller IoU thresholds.
- Most of these detections are false positives, and the recall is significantly lower for almost all classes.
- The average precision is also significantly lower in Rotated RetinaNet for almost all classes.

Confusion Matrices

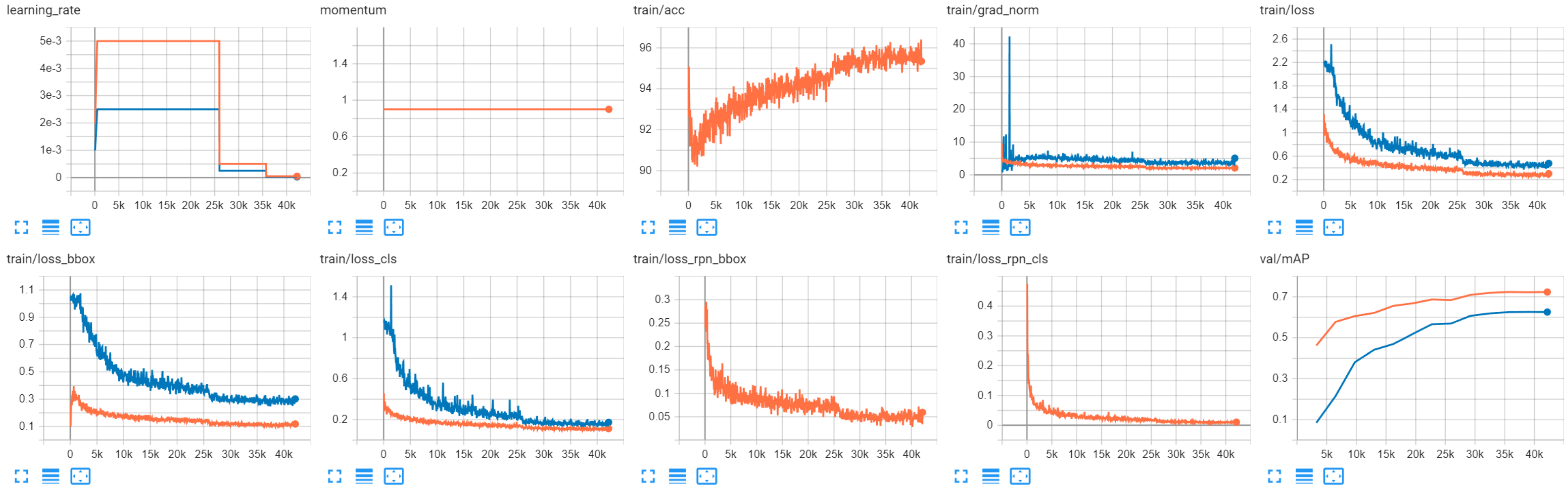




Confusion Matrices

- The confusion matrices were constructed by using a true positive IoU threshold of 0.5 and a score (probability) threshold of 0.3.
- The detection matrices are normalized row-wise (rows add up to 1) which hides the absolute numbers of the detections that we saw in the Metrics table.
- On the classification task, Oriented RCNN performs much better.
- The main misclassification sources in both models are
    - Mixing up courts, which is expected since their differences can be subtle.
    - Mixing up large and small vehicles, which is normal since some vehicles can be classified as either class
    - Classifying helicopters as planes, with the cause possibly being that there are much fewer helicopter instances in the training set (822) than planes (14,085), and that the objects have similar properties, for example elongated shape and materials.
- Rotated RetinaNet had a lot of trouble detecting bridges.
- The background row is heavily influenced by the distribution of instances, so it’s hard to make any statements about it.

Learning Curves



- Rotated RetinaNet
- Oriented RCNN

https://tensorboard.dev/experiment/ZJWKBf2jQ16ndAt060u3lg/

Learning Curves

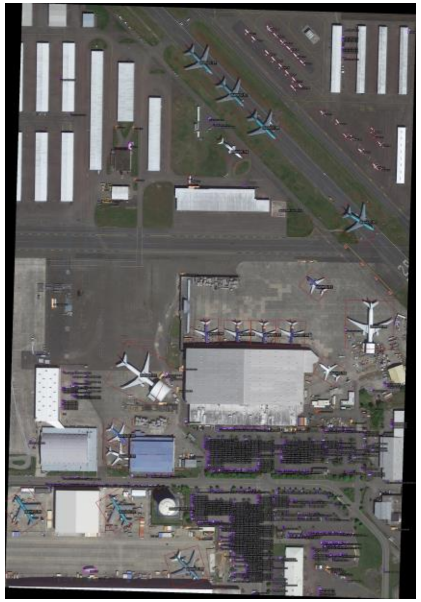
- Observe that the final bounding box and classification losses are consistently lower in Oriented RCNN, which is expected since it's a two-stage detector, while Rotated RetinaNet is a single stage detector.
- The maximum L2 norm of the gradients of Rotated RetinaNet rose to 42.24 at step 1400, therefore all parameters were multiplied by 35/42.24 (gradient clipping at 35). At the same step we see a jump in the classification loss.

Inference on the Original Images

- We perform inference on images of arbitrary size by cropping them into
  - 1024 × 1024 patches with step 824 (200 overlap),
  - as well using crop=1024//2 × 1024//2, step=824//2,
  - and crop=1024*2 × 1024*2, step=824*2,
- Patches are resized with bilinear interpolation before they are fed to the model
- The results are then merged by non-max suppression with an IoU threshold of 0.1.

Airport Example

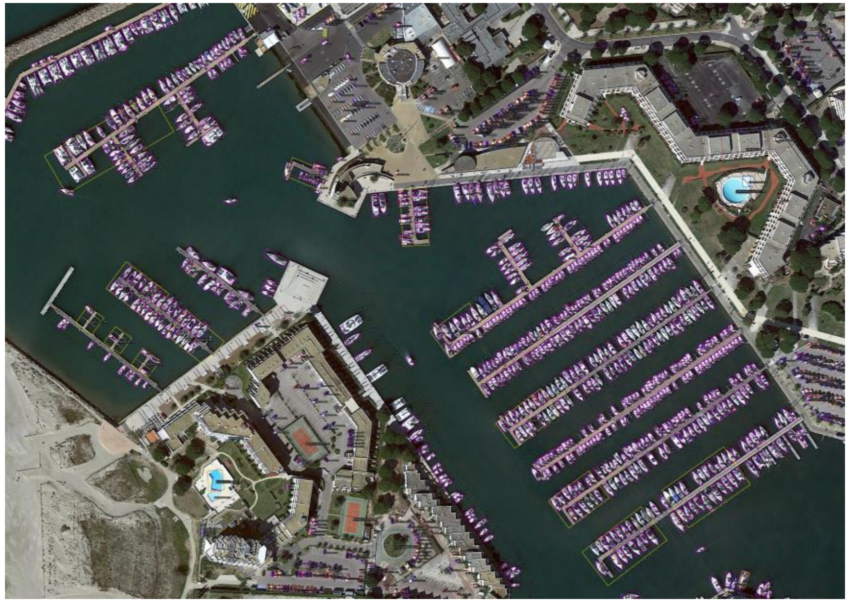
Rotated RetinaNet



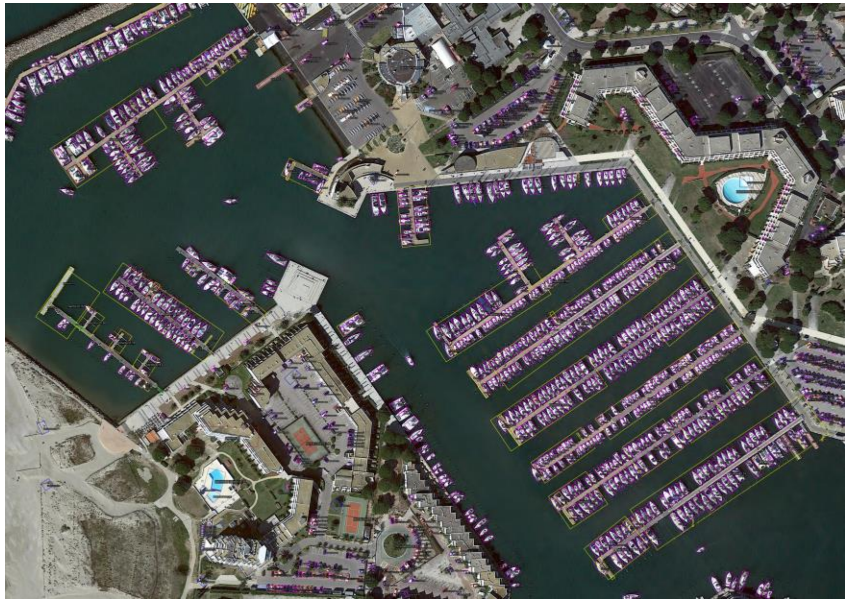
Oriented RCNN

Port Example

Rotated RetinaNet



Oriented RCNN



Parking Example

Rotated RetinaNet

Oriented RCNN

More predictions

- https://ntuagr-my.sharepoint.com/:f:/g/personal/konstantinospapadakis_ntua_gr/EhoFSVHIOVFGvjtszbnJlzoBfYHqU9DUWlhJ1_xswzcR0g?e=pCZiOX