1.Introduction

Aim : Evaluate the performance of polarmask on DOTA (Dataset for Object Detection in Aerial Images) dataset and compare with existing state-of-the-art models. Extend the research to including some of the ideas from existing models into polar mask to improve the segmentation.

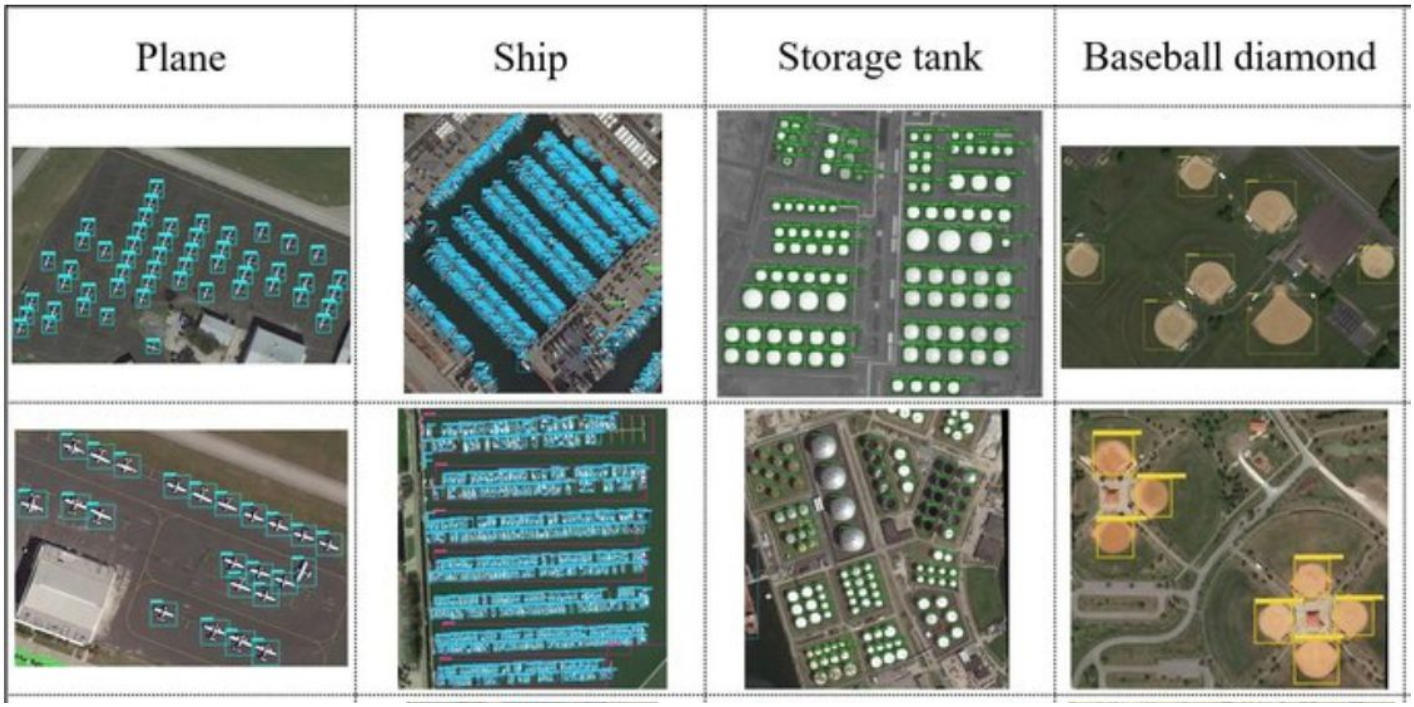
Plan of action:

a. Reimplement results of R^3 Det ++, FCOS, ROI transformer on DOTA dataset
b. Obtain results of plain polarmask on DOTA for comparison (instance segmentation)
c. Error Analysis of plain polarmask on DOTA (instance segmentation)
d. Make changes to modules of polarmask for oriented bounding box prediction.
e. Error analysis and improvements

2. Implementation

2.1 DOTA dataset

DOTA dataset, which is a satellite image dataset, has 2806 images where the annotations are given as 4 coordinate points of the oriented bounding box and it has 15 categories including plane,ship, basketball court etc. Within these 2806 images there are over 180 K instances which shows that there are a large number of instances in each image. Despite such complexity of the dataset, the state of the art models for the dataset is achieving over 0.8 mAP.



<Figure 1, DOTA dataset examples>

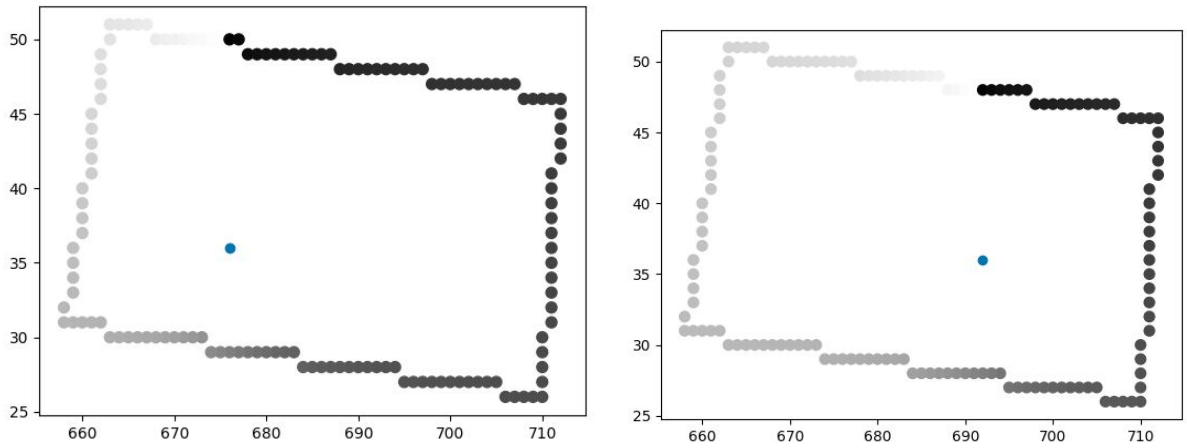
2.2 Data Preprocessing / Postprocessing

In comparison with polmask's implementation for HBB, the additional processing are the following,

1. Process annotation (x1,y1… x4,y4) to distance (top, right, bottom, left, angle)
2. Model output (top, right, bottom, left, angle) to oriented bounding box (x_ct, y_ct, w, h, angle)

2.1. (x1,y1… x4,y4) to distance (top, right, bottom, left, angle)

The way that these distances are generated are by making use of the contour of GT mask.

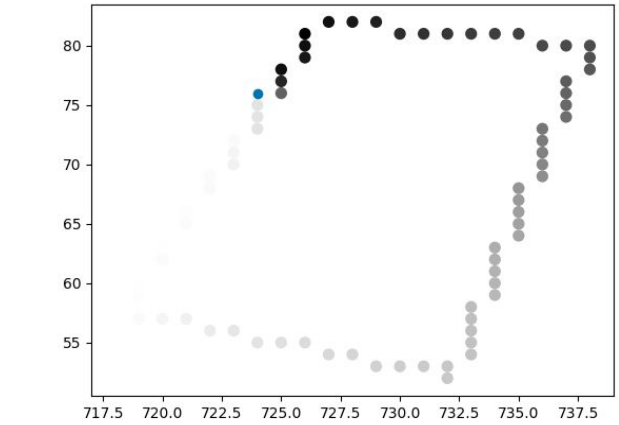


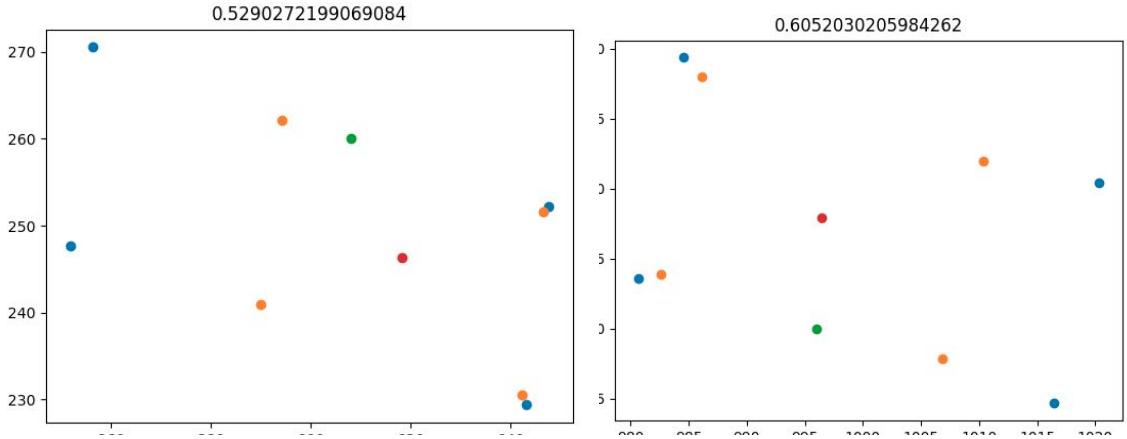
<Figure 2, Contour of oriented box for distance>

This is the contour of the bounding box and color representing the angle with respect to the subject pixel position. Clockwise movement along the contour represents angle increasing from 0 to 360; then we make use of the GT angle of the bbox to find the four rays. Hence the four distance (top, right, bottom, left, angle)

When the point is near the border the angle to be regressed becomes inaccurate

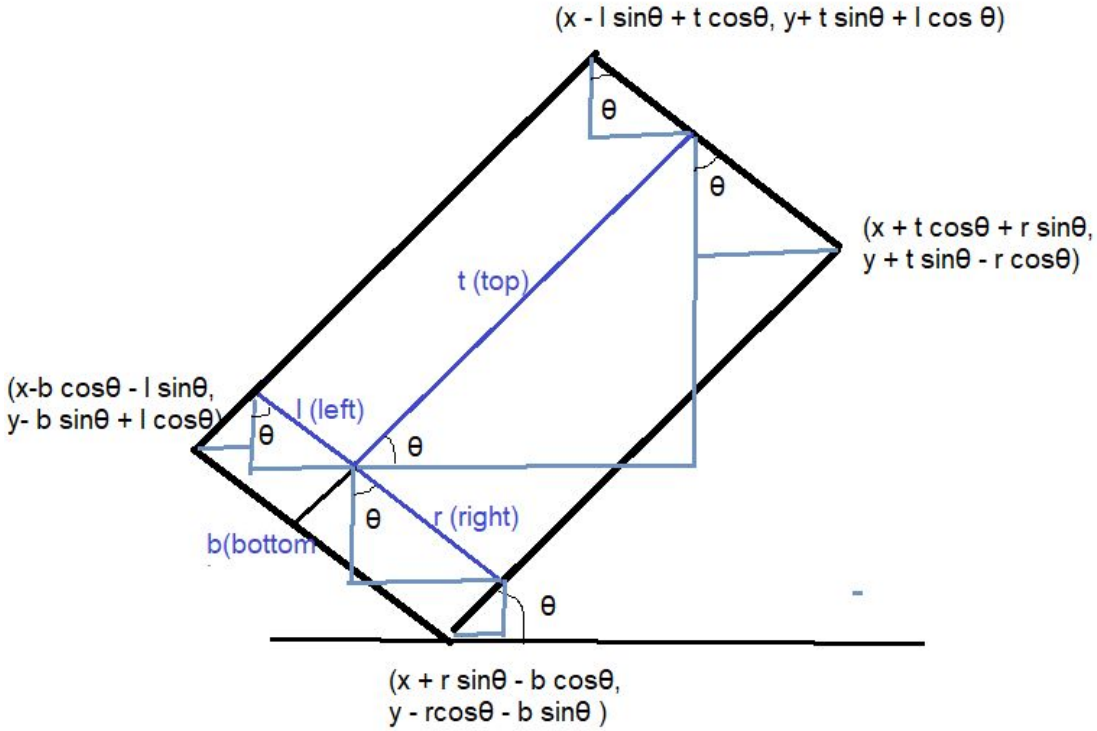
Furthermore there are cases where the contour of the box is sparse so that the rays are not detected. That would create error cases as below.





<Figure 3, Reconstruction when only 3 rays detected >

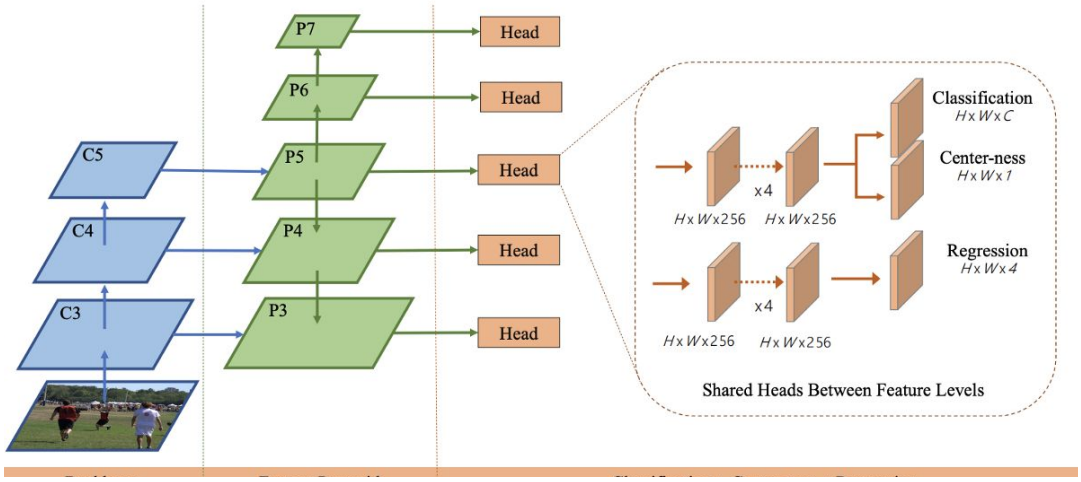
2.3 Distance (top, right,bottom, left, angle)-> polygon (x1,y1,..,x4,y4) ->obb (x_center, y_center, w, h, angle)



<Figure 4, distance to 8 coordinates (polygon)>

This change resulted in a skew IoU between ~ 0.9.

3. Architecture of model



<Figure 5, Architecture of FCOS>

As mentioned above, to make the model invariant to scale variation, FCOS makes use of FPN (feature pyramid network) as the backbone of the model. If we were to process multiple scaled images through other methods like

data augmentation or image pyramid, it takes a long time to process multiple scale images, and the memory requirements are too high. So in case where speed is not important, one can employ image pyramids or one can use such methodology only during inference for maximum accuracy. Otherwise it is quite conventional to incorporate a pyramid of features to use for object detection.

The model will output 3 types of output: namely, centeredness, class probability and regression. We can think of class probability as a vector with length K and regression as vector with length 4 (top,right,bottom,left). Each position in the multiple feature map will produce these 3 outputs. And the modification I have made is to change the length of the regression vector from 4 to 5; that is to append a target of angle as well.

Output of model will have shapes as below:
There are 5 feature levels and the (w_i, h_i) represents the dimension of feature level.

- Class score = List(feature level, N, 15, w_i , h_i)
- Bbox pred = List(feature level, N , 5, w_i, h_i)
- Centeredness = List (feature level, N , w_i, h_i)

This output is reduced into detection boxes through rotated NMS and its labels based on the positive, negative IoU threshold value and other hyperparameters (attached in appendix).

3.2 Loss and Target

The offset of our model is constructed as figure 12. Ideally we want each of the values to be equal to the ground truth; due to the nature of logarithm the closer the value is to 1 the closer the value will be to 0. This offset is consistent with the RoI tranformer's rotated region proposal network offset.

$$t_x^\star = log(\frac{x_{ct}^\star - x_{ct}}{w*}) \qquad t_y^\star = log(\frac{y_{ct}^\star - y_{ct}}{h^\star})$$

$$t_w^\star = log(\frac{w^\star}{w}) \qquad t_h^\star = log(\frac{h^\star}{h})$$

$$t_\theta^\star = \frac{\theta^\star - \theta'}{\pi/2}$$

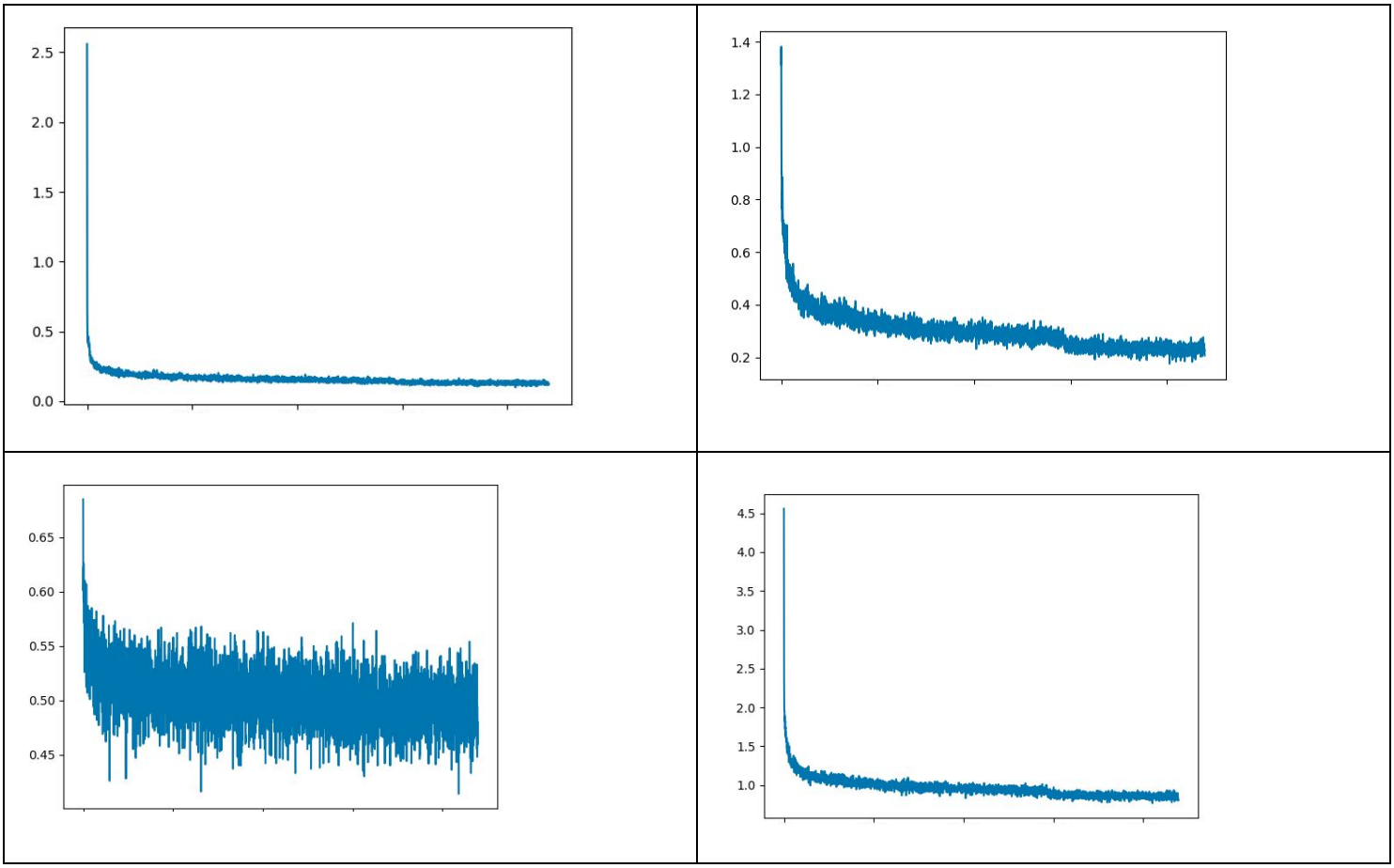
<Figure 6 , Offset of regression branch>

Here variables with star superscript indicate the ground truth while the variables with superscript ‘ are the predicted values. By making use of smooth l1 loss, the model learns to make these offset equal to 0 implying the convergence to the ground truth.

4. Results

4.1. Loss graph

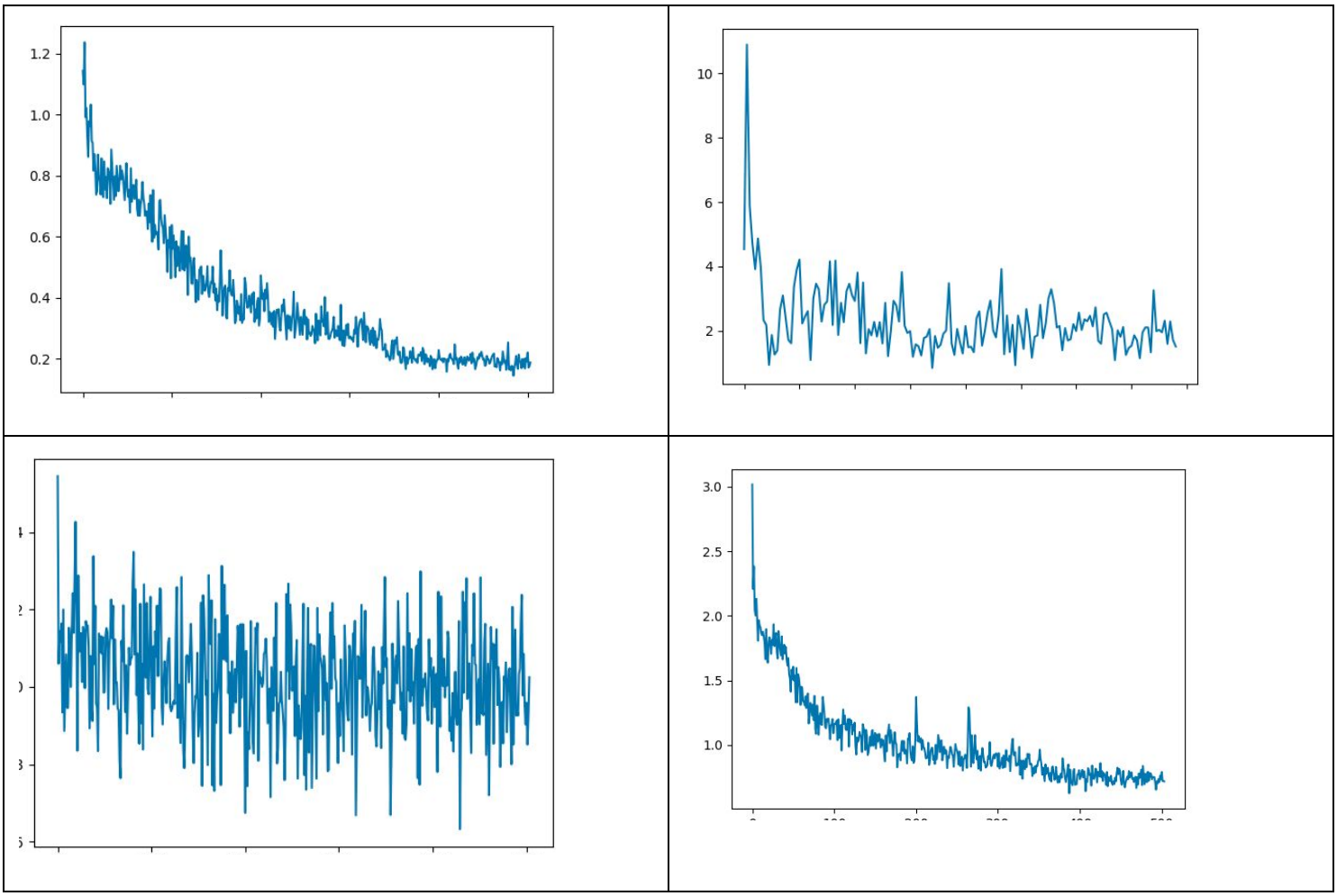
- HBB Polarmask on COCO dataset



<Figure , Mask, class, centeredness loss in clockwise direction from left top >

4.2 Training result

- OBB Polarmask on DOTA dataset



<Figure , regression, class, centeredness loss in clockwise direction from left top >

3.3. Test result

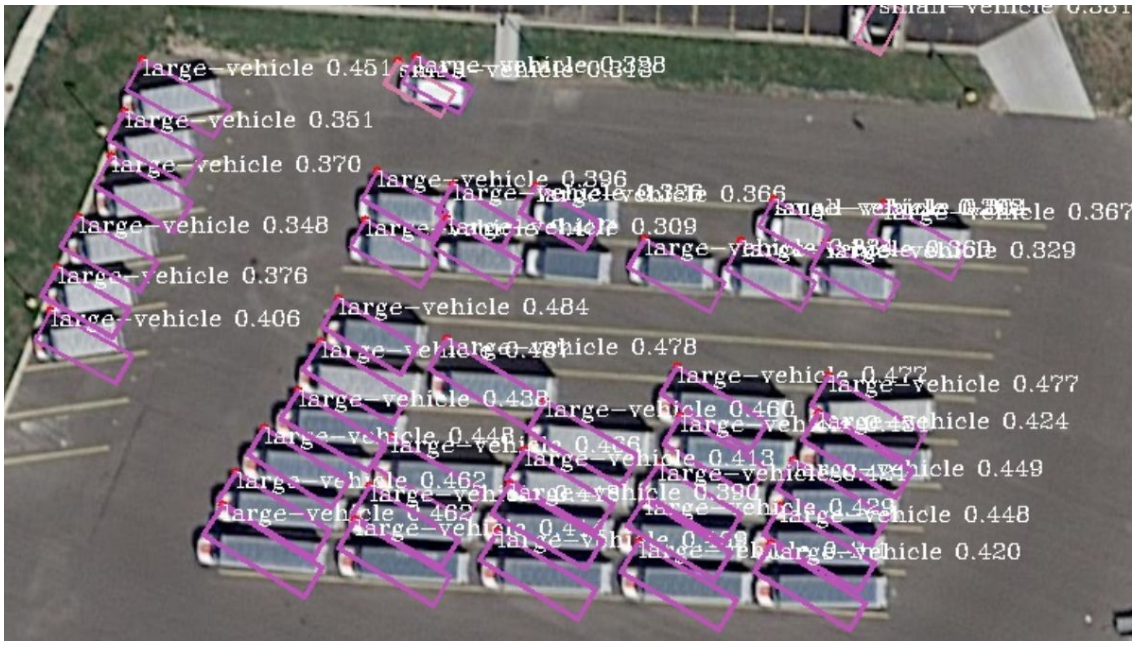
mAP	Beta	Angle Target	Weight (cls, bbox)	Number of rays	Regression weight
0.28	1	theta/(pi/2)	(5,1)	4	(1,1,1,1,1)
0.27	1	theta/(pi)	(5,1)	4	(1,1,1,1,1)
0.504	1	theta/(pi)	(5,2)	4	(10,10,5,5,1)
0.48	0.5	theta/(pi/2)	(5,2)	3	(10,10,5,5,1)
0.49	1	theta/(pi/2)	(5,3)	3	(10,10,5,5,1)
0.55	0.5	theta/(pi/2)	(5,3)	4	(10,10,5,5,1)

3.3.1. Top 1 model performance analysis

- **mAP**: 0.5503655127508457
- **ap of each class**: plane:0.8607046927853989, baseball-diamond:0.5568282283792001, bridge:0.2681579311381596, ground-track-field:0.4670096834616287, small-vehicle:0.6397068245709991, large-vehicle:0.495478672355514, ship:0.595660700854517, tennis-court:0.9054610913927454, basketball-court:0.6683319591111994, storage-tank:0.7998760995513182, soccer-ball-field:0.36889937350000385, roundabout:0.5127014830735858, harbor:0.41323499141009273, swimming-pool:0.29802548378444876, helicopter:0.405405475893871

-As we can see from the class mAP, large objects like plane or tennis-courts are well detected while small objects like small-vehicle or cluttered objects like ships are not performing well.

4. Visualisation



4 Further Improvement

As mentioned above, the major challenges to be solved from this point is to deal with small objects in a cluttered scene and fix the problem of angle regression. Referring to the models discussed before, possible methods for improvements would be as such:

1. Refinement of predictions based on the ideas of R^3 det +++
    - Refinement has been proven to be effective by different types of refined object detectors (cascade RCNN, RefineDet)
2. Pixel attention branch on the feature maps of output of FPN neck based on ideas of SCRDet
    - This is shown to be especially effective to deal with images with cluttered instances.
3. Weighted regression loss
    - Similar to focal loss, we aim to put more emphasis on the angle regression because it is a crucial factor in the prediction of oriented boxes; small change in angle leads to large drop in skew IoU.
    - Therefore adding weight to the smooth L1 loss in a way to regularize angle prediction can be an effective way to improve performance.

Appendix : Configuration for model

Model was implemented using mmdetection and mmcv library which are built on pytorch platform.

```
model = dict(
    type='PolarMask',
    pretrained='open-mmlab://resnet50_caffe',
    backbone=dict(
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        frozen_stages=1,
        norm_cfg=dict(type='BN', requires_grad=False),
        style='caffe'),
    neck=dict(
        type='FPN',
        in_channels=[256, 512, 1024, 2048],
        out_channels=256,
        start_level=1,
        add_extra_convs=True,
        extra_convs_on_inputs=False,  # use P5
        num_outs=5,
        relu_before_extra_convs=True),
    bbox_head=dict(
        type='PolarMask_Head',
        num_classes=16,
        in_channels=256,
        stacked_convs=4,
        feat_channels=256,
        strides=[8, 16, 32, 64, 128],
```

```
        loss_cls=dict(
            type='FocalLoss',
            use_sigmoid=True,
            gamma=2.0,
            alpha=0.25,
            loss_weight=3.0),
        loss_bbox=dict(type='SmoothL1Loss', beta=0.5, loss_weight=5),
        loss_centerness=dict(
            type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0)))
# training and testing settings
train_cfg = dict(
    assigner=dict(
        type='MaxIoUAssignerRbbox',
        pos_iou_thr=0.5,
        neg_iou_thr=0.4,
        min_pos_iou=0,
        ignore_iof_thr=-1),
    allowed_border=-1,
    pos_weight=-1,
    debug=False)
test_cfg = dict(
    nms_pre=1000,
    min_bbox_size=0,
    score_thr=0.05,
    nms=dict(type='py_cpu_nms_poly_fast', iou_thr=0.1),
    max_per_img=100)
# dataset settings
dataset_type = 'DOTADataset'
data_root = 'data/dota1_aug/'
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
data = dict(
    imgs_per_gpu=2,
```

```
workers_per_gpu=2,
train=dict(
    type=dataset_type,
    ann_file=data_root + 'trainval1024/DOTA_trainval1024.json',
    img_prefix=data_root + 'trainval1024/images/',
    img_scale=(1024, 1024),
    img_norm_cfg=img_norm_cfg,
    size_divisor=32,
    flip_ratio=0.5,
    with_mask=True,
    with_crowd=True,
    with_label=True),
val=dict(
    type=dataset_type,
    ann_file=data_root + 'trainval1024/DOTA_trainval1024.json',
    img_prefix=data_root + 'trainval1024/images',
    img_scale=(1024, 1024),
    img_norm_cfg=img_norm_cfg,
    size_divisor=32,
    flip_ratio=0,
    with_mask=True,
    with_crowd=True,
    with_label=True),
test=dict(
    type=dataset_type,
    ann_file=data_root + 'test1024/DOTA_test1024.json',
    img_prefix=data_root + 'test1024/images',
    img_scale=(1024, 1024),
    img_norm_cfg=img_norm_cfg,
    size_divisor=32,
    flip_ratio=0,
    with_mask=True,
    with_crowd=True,
```

```
        with_label=True,
        test_mode=True))

# optimizer
lr_ratio = 1

optimizer = dict(
    type='SGD',
    lr=0.01 * lr_ratio,
    momentum=0.9,
    weight_decay=0.0001,
    paramwise_options=dict(bias_lr_mult=2., bias_decay_mult=0.))
optimizer_config = dict(grad_clip=dict(max_norm=20, norm_type=2))
# learning policy
lr_config = dict(
    policy='step',
    warmup='linear',
    warmup_iters=500,
    warmup_ratio=1.0 / 3 / lr_ratio,
    step=[8, 11])
checkpoint_config = dict(interval=1)
# yapf:disable
log_config = dict(
    interval=20,
    hooks=[
        dict(type='TextLoggerHook'),
        # dict(type='TensorboardLoggerHook')
    ])
# yapf:enable
# runtime settings
total_epochs = 12
device_ids = range(4)
dist_params = dict(backend='nccl')
```

```
log_level = 'INFO'
work_dir = './work_dirs/trash'
load_from = None
resume_from = None
workflow = [('train', 1)]
```