

به نام خدا



دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

**درس شبکه‌های عصبی و یادگیری عمیق  
تمرین سوم**

علیرضا نجفی مطیعی	نام و نام خانوادگی	پرسش ۱
810100224	شماره دانشجویی	
پوریا ذره پرور قوچانی نژاد	نام و نام خانوادگی	پرسش ۲
۸۱۰۱۰۲۱۴۱	شماره دانشجویی	
<b>۱۴۰۴.۰۲.۰۲۱</b>	مهلت ارسال پاسخ	

## فهرست

1.....	پرسش 1. سگمنتیشن تصاویر شهری
1.....	1. توصیف مدل ارائه شده
2.....	2. آمادهسازی مجموعه داده
4.....	3. بهینهساز، متريکها وتابع هزينه:
6.....	4. پيادهسازی مدل
8.....	5. آموزش مدل
10.....	6. ارزیابی مدل
12.....	پرسش 2. Oriented R-CNN برای تشخیص اشیا

## شکل‌ها

شکل 1. عنوان تصویر نمونه ..... Error! Bookmark not defined.

## جدول‌ها

جدول 1. عنوان جدول نمونه ..... Error! Bookmark not defined.

## پرسش ۱. سگمنتیشن تصاویر شهری

### ۱-۱. توصیف مدل ارائه شده

در این مقاله به منظور کاربرد سریعتر از پردازش real-time و پردازش سریعتر نسبت به مدل‌های encoder-decoder یک مدل Fast-SCNN برروی تصاویر با رزولوشن بالا( $1024*2048\text{px}$ ) که برای محاسبات بهینه و سسیتم‌های با مموری کم مناسب است، ارائه شده است. این مدل به نوعی یک مدل two-branch از دو شاخه کلی استفاده می‌کند بگونه‌ای که یک شاخه عمیق تر برای جمع‌آوری اطلاعات کلی در رزولوشن پایین استفاده می‌شود و یک شاخه کم عمق تر برای یادگیری جزئیات فضایی در رزولوشن کامل استفاده می‌شود. نتیجه نهایی segmentation از ترکیب این دو بدست می‌آید. تفاوت این مدل یک مدل learning to downsample همزمان اطلاعات فضایی two-branch عادی در این است که در این مدل بخش learning to downsample همچنان اطلاعات دو شاخه‌ای و لایه‌های اولیه و زمینه‌های کلی را encode می‌کند و درواقع این مدل ترکیبی از مدل‌های دو شاخه‌ای و همچنین encoder-decoder است. درواقع لایه‌های ورودی Deep Convolutional Neural Network برای استخراج ویژگی‌های سطح پایین می‌شوند و محاسبات لایه‌های ورودی در رویکرد دو لایه‌ای توضیح داده شده به اشتراک گذاشته می‌شوند. در نهایت این مدل کانولوشن‌های مجزای عمیق و بهینه را اتخاذ می‌کند. ساختار کلی این مدل بصورت زیر است که بخش learning to down-sample نقش استخراج ویژگی‌های سطح پایین و همچنین جزئیات فضایی را دارد. بلاک‌های مورد استفاده در این مدل در بخش پیاده‌سازی مدل توضیح داده خواهند شد.

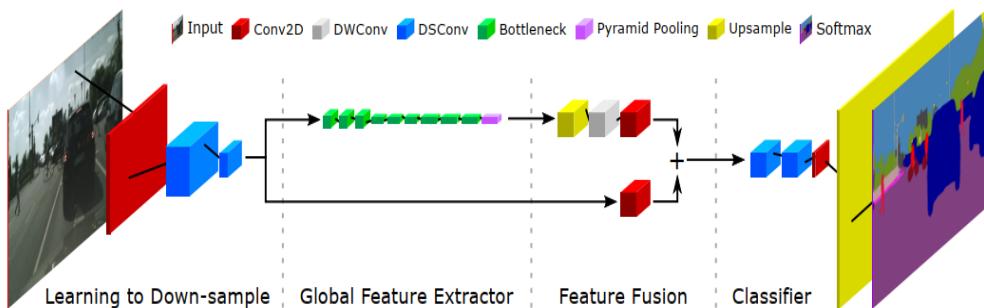


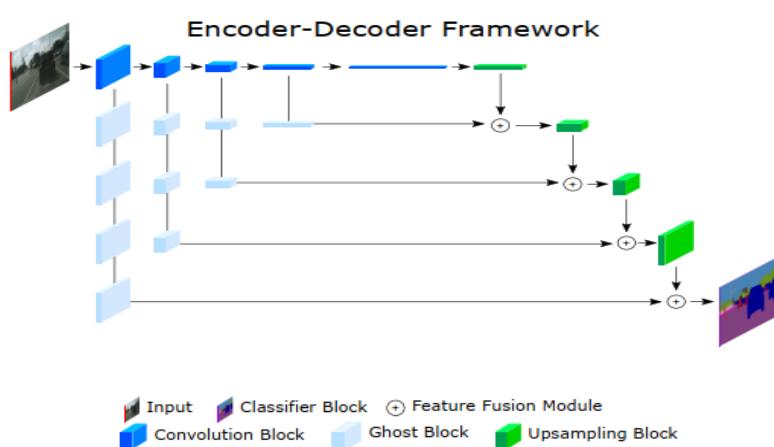
Figure 1. Fast-SCNN shares the computations between two branches (encoder) to build a above real-time semantic segmentation network.

### شکل ۱. ساختار کلی مدل

مدل‌های encoder-decoder بجای استفاده از مازول learning to down-sample از skip connections بصورت کلاسیک استفاده می‌کنند که در این حالت خروجی یک لایه به خروجی یک لایه عمیق‌تر متصل می‌شود. در حالت کلی ساختار یک مدل encoder-decoder بگونه‌ای است که یک مسیر وجود دارد که در هر رزولوشن از طریق تعدادی skip connection بصورت مستقیم به متصل می‌شود که متفاوت از رویکرد موجود در مقاله است که درواقع ترکیبی از مدل‌های دو شاخه‌ای دو همچنین مدل‌های کلاسیک است. از دیگر تفاوت‌های ساختاری این مدل با یک مدل encoder-decoder در این است که در اینجا فیچرهای بدست‌آمده از رزولوشن بالا و پایین را باهم جمع می‌کنیم اما مدل کلاسیک ویژگی‌های انکودر و دیکودر را به هم پیوند می‌دهد تا دقت را بالا ببرد. همچنین در متن مقاله اشاره شده که پارامترهای این مدل در حدود ۱.۱۱ میلیون پارامتر هستند در صورتیکه این مقدار برای مدل U-net چیزی در حدود ۲۳ میلیون پارامتر دارد.

از نظر عملکردی هرچند مدل Fast-SCNN سریعتر خواهد بود اما با توجه به لایه‌های عمیق‌تر و Skip Connection های موجود در مدل U-Net این مدل دقیق‌تر خواهد داشت. همچنین طبق گفته مقاله مقدار mIoU بدست آمده توسط این مدل ۶۸٪ است که از میزان این فاکتور برای مدل U-Net برای تصویرهای پزشکی (چیزی در حدود ۹۲٪) پایین‌تر است. همچنین طبق گفته مقاله برای این مدل Pre-training تفاوت چندانی ایجاد نمی‌کند و لازم نیست اما برای مدل U-Net معمولاً نیاز به داریم. بنابراین در حالت کلی با توجه به سرعت بالاتر و محاسبات بهینه استفاده از Fast-SCNN برای کاربردهای real-time مناسب‌تر است اما برای کاربردهایی که نیاز به دقیق‌تری دارند استفاده از U-Net بهتر است.

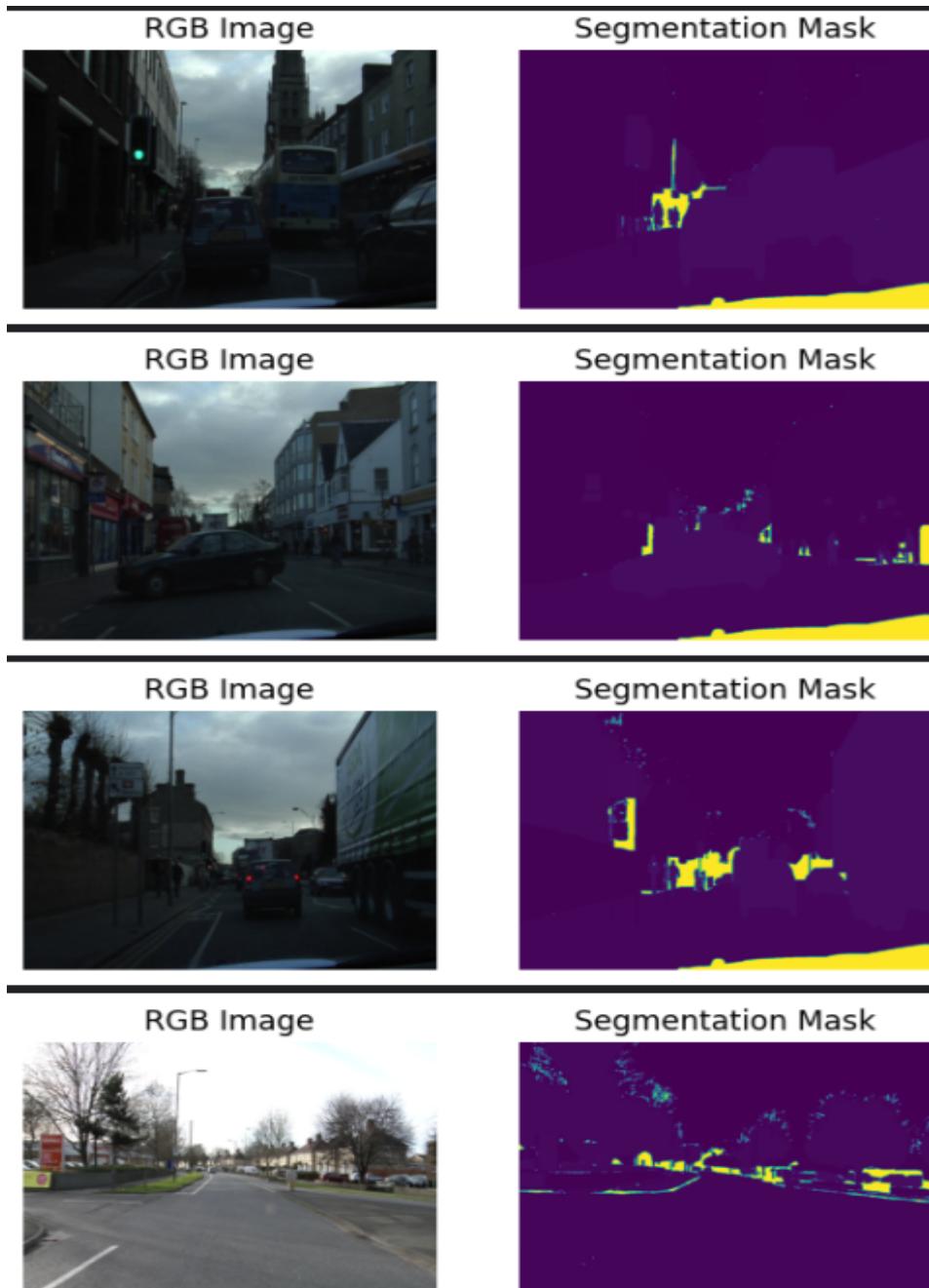
همچنین یک نمونه از معماری ساختار encoder-decoder به فرم زیر است:



شکل 2. ساختار کلی encoder-decoder

## 2-1. آماده‌سازی مجموعه داده

در این دیتابیس داده‌های CamVid\_RGB را به عنوان تصاویر اصلی و داده‌های CamVidColor11 به عنوان mask استفاده می‌کنیم. چند نمونه از تصاویر اصلی به همراه ماسک آنها در زیر آمده است:



شکل 3. نمونه‌ای از تصاویر به همراه ماسک آن‌ها

همچنین تعداد داده‌های آموزش، اعتبارسنجی و تست را بدست می‌آوریم که بصورت زیر است:

```
Dataset split counts:  
Train: 367 images  
Val: 101 images  
Test: 233 images
```

شکل 4. تعداد داده‌ها

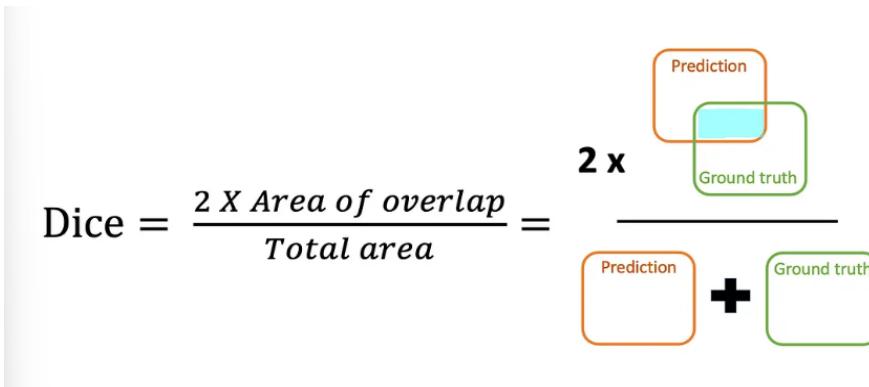
### 3-1. بهینه‌ساز، متریک‌ها وتابع هزینه:

DiceCoefficient و IoU Score دو متریکی هستند که برای ارزیابی یک segmenatation از آن‌ها استفاده می‌شود.

:Dice Coefficient به نوعی معادل F1-Score برای مسائل segmentation است که از معیارهای precision و recall محاسبه می‌شود و درواقع میانگین هارمونیکی از precision و recall است. به عبارت دیگر این معیار از دوبرابر همپوشانی میان سگمنتیشن پیشビینی شده و واقعیت تقسیم بر میزان کل پیکسل‌های این دو بدست می‌آید که رابطه آن بصورت زیر است(با ماتریس آشفتگی در تمرین‌های قبلی و درس‌های پایه‌ای تر آشنا شده‌ایم به همین دلیل از توضیح مجدد پارامترهای آن خودداری می‌کنیم):

$$\text{Dice Coefficient} = \frac{2TP}{2TP + FP + FN}$$

همچنین دید تصویری محاسبه این معیار بصورت زیر است:

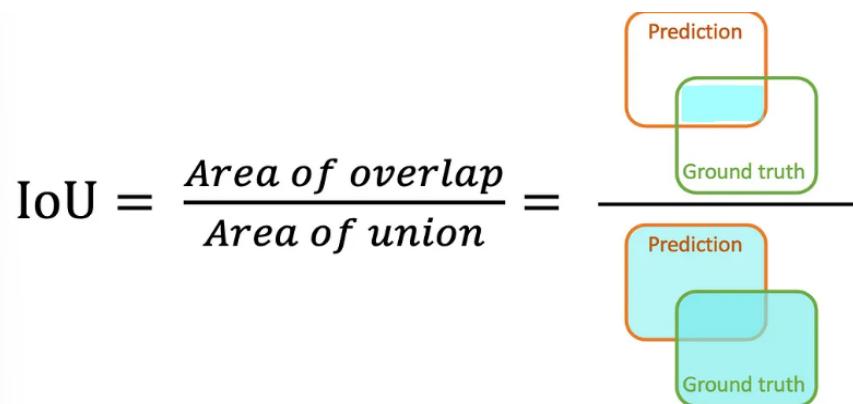


شکل 5. محاسبه معیار Dice Coefficient

Jaccard Index (IoU) که با Intersection over Union (IoU) نیز شناخته می‌شود درواقع مساحت همپوشانی بین سگمنتیشن پیشビینی شده و واقعیت را نسبت به مساحت کل این دو می‌سنجد و نحوه محاسبه آن بصورت زیر است:

$$IoU = \frac{TP}{TP + FP + FN}$$

همچنین دید تصویری محاسبه این معیار بصورت زیر است:



## شکل 6. محاسبه معیار IoU

از آنجایی که در واقع تعداد کل پیشینی‌های درست نسبت به کل پیشینی‌هاست و پیشینی درست زمینه تصویر و سگمنتیشن نادرست می‌تواند باز هم Accuracy بالایی ثبت کند بهتر است از دو معیار ذکر شده در ارزیابی این موارد بگیریم.

مطابق با فرمول‌های نوشته شده دو کلاس برای این معیارها تعریف می‌کنیم و در مرحله بعد در حین آموزش شبکه برای بهینه‌سازی آن به همراه Accuracy از آن استفاده خواهیم کرد. در مورد بهینه‌ساز و تابع هزینه نیز در بخش‌های بعد توضیح داده خواهد شد.

```
class MeanIoUImplemented(tf.keras.metrics.Metric):
    def __init__(self, num_classes, name="mean_iou_implemented", **kwargs):
        super().__init__(name=name, **kwargs)
        self.num_classes = num_classes
        self.total_intersection = self.add_weight(name="total_intersection",
                                                initializer="zeros", shape=(num_classes,))
        self.total_union = self.add_weight(name="total_union",
                                           initializer="zeros", shape=(num_classes,))

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_pred = tf.argmax(y_pred, axis=-1)
        y_true = tf.cast(y_true, tf.int32)
        y_pred = tf.cast(y_pred, tf.int32)

        cm = tf.math.confusion_matrix(tf.reshape(y_true, [-1]), tf.
                                      reshape(y_pred, [-1]), self.num_classes)
        intersection = tf.linalg.diag_part(cm)
        union = tf.reduce_sum(cm, axis=1) + tf.reduce_sum(cm, axis=0) - intersection

        self.total_intersection.assign_add(intersection)
        self.total_union.assign_add(union)

    def result(self):
        iou = (self.total_intersection + 1e-7) / (self.total_union + 1e-7)
        return tf.reduce_mean(iou)

    def reset_state(self):
        self.total_intersection.assign(tf.zeros_like(self.total_intersection))
        self.total_union.assign(tf.zeros_like(self.total_union))
```

```

class DiceCoefficientImplemented(tf.keras.metrics.Metric):
    def __init__(self, num_classes, name="dice_coefficient_implemented", **kwargs):
        super().__init__(name=name, **kwargs)
        self.num_classes = num_classes
        self.total_intersection = self.add_weight(name="total_intersection",
                                                initializer="zeros", shape=(num_classes,))
        self.total_true = self.add_weight(name="total_true", initializer="zeros",
                                         shape=(num_classes,))
        self.total_pred = self.add_weight(name="total_pred", initializer="zeros",
                                         shape=(num_classes,))

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_pred = tf.argmax(y_pred, axis=-1)
        y_true = tf.cast(y_true, tf.int32)
        y_pred = tf.cast(y_pred, tf.int32)

        cm = tf.math.confusion_matrix(tf.reshape(y_true, [-1]), tf.
                                      reshape(y_pred, [-1]), self.num_classes)
        intersection = tf.linalg.diag_part(cm)
        true_sum = tf.reduce_sum(cm, axis=1)
        pred_sum = tf.reduce_sum(cm, axis=0)

        self.total_intersection.assign_add(intersection)
        self.total_true.assign_add(true_sum)
        self.total_pred.assign_add(pred_sum)

    def result(self):
        dice = (2.0 * self.total_intersection + 1e-7) / (self.total_true + self.total_pred + 1e-7)
        return tf.reduce_mean(dice)

    def reset_state(self):
        self.total_intersection.assign(tf.zeros_like(self.total_intersection))
        self.total_true.assign(tf.zeros_like(self.total_true))
        self.total_pred.assign(tf.zeros_like(self.total_pred))

```

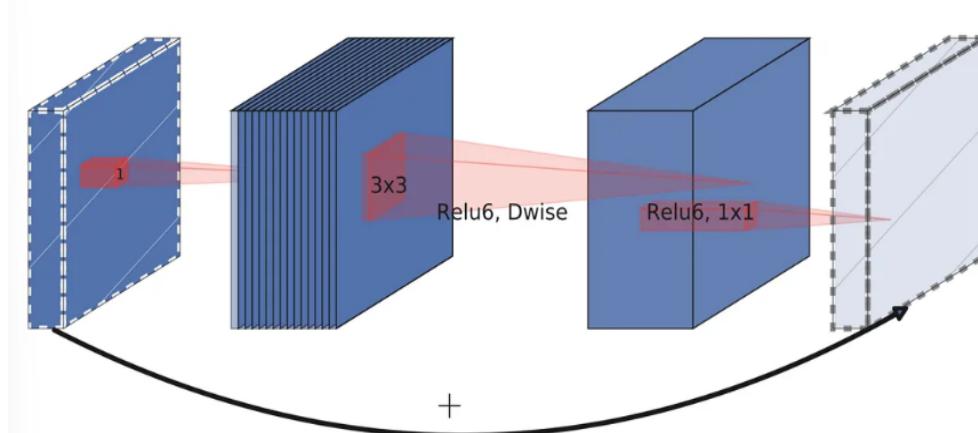
## 4-1. پیاده‌سازی مدل

: برای توضیح این نوع از convolution Depthwise Separable Convolution و depth-wise convolution point-wise convolution بپردازیم. همچنین فرض می‌گیریم تعداد چنل‌های تصویر برابر با  $M$ ، سایز تصویر  $D_F \times D_F$ ، سایز کرنل‌ها  $D_K \times D_K \times M$  است. تعداد فیلترها  $N$  و سایز خروجی  $D_P \times D_P \times M$  می‌باشد. در depth-wise convolution بخلاف CNN استاندارد که در آن در هر مرحله Convolution برای تمام  $M$  چنل موجود در تصویر اتفاق می‌افتد، فرآیند convolution هر بار برای یک چنل انجام می‌شود بنابراین سایز کرنل در اینجا  $1 \times 1 \times D_K \times D_K$  می‌باشد. برای  $M$  چنل، عدد از این کرنل‌ها مورد نیاز خواهد بود. بنابراین تعداد کل ضربهای مورد نیاز در این حالت برابر با  $M \times D_K^2 \times D_P^2$  خواهد بود. در point-wise convolution یک عملیات  $1 \times 1$  بروی  $D_P \times D_P \times N$  چنل صورت می‌پذیرد. با فرض نیاز به استفاده از  $N$  عدد از این عملیات‌ها سایز خروجی  $D_P \times D_P \times N$  خواهد بود. بنابراین تعداد کل ضربهای در این فرآیند  $M \times D_P^2 \times N$  می‌باشد. برای کل فرآیند depth-wise separable convolution که ترکیب این دو است باید دو مقدار قبلی را جمع کنیم که در نهایت تعداد کل ضربهای خواهد بود همچنین این تعداد برای حالت استاندارد  $M \times (D_K^2 + N) \times D_P^2$  است بنابراین نسبت کلی  $\frac{1}{N} + \frac{1}{D_K^2}$  می‌باشد. یعنی برای مثال برای  $N=100$  و  $D_K = 512$  این نسبت در حدود یک صدم خواهد بود که یعنی تعداد کل ضربهای در depth-wise separable convolution در حدود 0.01 حالت استاندارد است.

: این بلاک با دو فرض زیر طراحی شده است و دو فرض زیر طراحی لایه کانولوشنی جدید را هدایت می کند :

- (الف) نقشه های feature ها می توانند در زیر فضاهای کم بعد کدگذاری شوند
- (ب) فعال سازی های غیر خطی با وجود توانایی هایشان در افزایش پیچیدگی نمایشی، منجر به از دست رفتن اطلاعات می شوند.

این لایه یک تنسور کم بعد با  $k$  چنل را دریافت می کند و سه convolution مجزا انجام می دهد. در ابتدا یک point-wise convolution برای گسترش feature map کم بعد به به فضایی با ابعاد بالاتر و مناسب برای فعال سازی های غیر خطی انجام می شود و پس آن از ReLU استفاده می شود. در این مرحله expansion factor 3 برابر با  $t$  منجر به  $tk$  چنل می شود. در مرحله بعد یک depth-wise convolution با کرنل 3 در 3 انجام می پذیرد و به دنبال آن از فعال ساز ReLU استفاده می شود که به فیلترینگ مکانی تنسور با ابعاد بالاتر منجر می شود. در مرحله آخر feature map فیلتر شده مکانی با استفاده از یک point-wise convolution دیگر به یک زیر فضای کم بعد تصویر می شود. این تصویر کردن به تنها یکی موجب از دست رفتن اطلاعات می شود بنابراین مهم است کهتابع فعال ساز در مرحله آخر از نوع خطی باشد. زمانی که feature map اولیه و نهایی ابعاد یکسانی دارند یک اتصال باقی مانده برای کمک به جریان gradian در طول backpropagation اضافه می شود.



شکل 7. وضعیت feature map در این بلاک

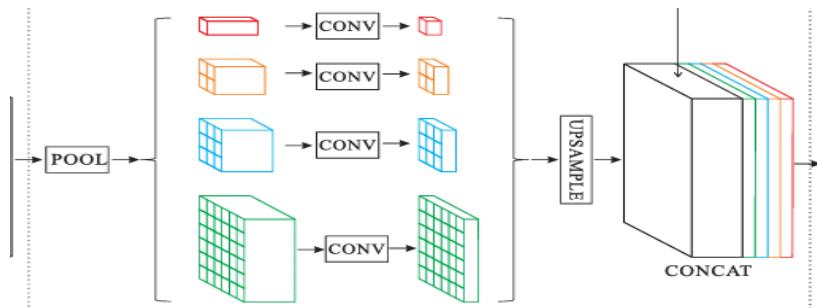
Input	Operator	Output
$h \times w \times k$	$1 \times 1 \text{ conv2d}, \text{ReLU6}$	$h \times w \times (tk)$
$h \times w \times tk$	$3 \times 3 \text{ dwise } s=s, \text{ReLU6}$	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear $1 \times 1 \text{ conv2d}$	$\frac{h}{s} \times \frac{w}{s} \times k'$

شکل 8. وضعیت ورودی و خروجی در این بلاک

: Pyramid Pooling Module

در یک شبکه عصبی عمیق اندازه receptive field می تواند بصورت تقریبی نشان دهد که ما چه مقدار از اطلاعات زمینه ای استفاده می کنیم. اگرچه از نظر تئوری receptive field معمولاً از تصویر ورودی بزرگتر است اما بصورت تجربی ثابت شده است در واقعیت این میزان به خصوص در لایه های سطح بالا بسیار کمتر

است و باعث می‌شود بسیاری از شبکه‌ها به اندازه کافی اطلاعات اولیه دریافت نکنند. Pyramid Pooling مازولی برای semantic segmentation است که به عنوان یک global contextual Module می‌کند. این مازول شامل اطلاعاتی با مقیاس‌های متفاوت است و از زیرناحیه‌ای به زیرناحیه دیگر متفاوت است. این مازول feature های چهار مقیاس هرمی مختلف را با هم ترکیب می‌کند. برای حفظ وزن feature ها یک لایه کانولوشنی  $1 \times 1$  بعد از هر pooling استفاده شده که بعد را با فرض بعد اولیه  $N$  برای هرم،  $\frac{1}{N}$  upsample، bilinear interpolation و به روش concatenate می‌شوند. در نهایت لول‌های مختلف feature ها به عنوان خروجی نهایی این مازول به یکدیگر می‌شوند.



شکل 9. ساختار کلی PPM

پس از پیاده‌سازی مدل مطابق با مقاله داده شده مطابق با انتظار پارامترهای مدل بصورت زیر خواهد بود. در مقاله نیز تعداد پارامترهای قابل آموزش 1.1 میلیون پارامتر گزارش شده بود. همچنین در این بخش چون سایز تصاویر دیتابست از  $2048 \times 1024$  بسیار کوچک‌تر است ورودی مدل را  $512 \times 512$  درنظر گرفتیم.

Total params:	1,125,916	(4.30 MB)
Trainable params:	1,103,260	(4.21 MB)
Non-trainable params:	22,656	(88.50 KB)

شکل 10. تعداد پارامترهای مدل ایجاد شده

## 5-1. آموزش مدل

در اینجا بجای استفاده از optimizer SGD نامبرده در مقاله که Adam optimizer است از learning\_rate گفته شده در مقاله عدد نسبتاً بزرگی بود ما نیز از 0.004 شروع می‌کنیم و با استفاده از callback با جلو رفتن در فرایند یادگیری این مقدار را با فاکتور 0.6 کاهش می‌دهیم. همچنین بمنظور کاهش overfit optimizer از weight\_decay با میزان 0.0001 نیز استفاده می‌کنیم.

در اینجا ازتابع هزینه dice loss که رابطه زیر را با Dice Coefficient دارد همزمان به همراه tf.keras.losses.sparse\_categorical\_crossentropy استفاده می‌کنیم:

$$Dice Loss = 1 - Dice Coefficient$$

$$Loss = \alpha * crossentropy + (1 - \alpha) * Dice Loss, \alpha = 0.4$$

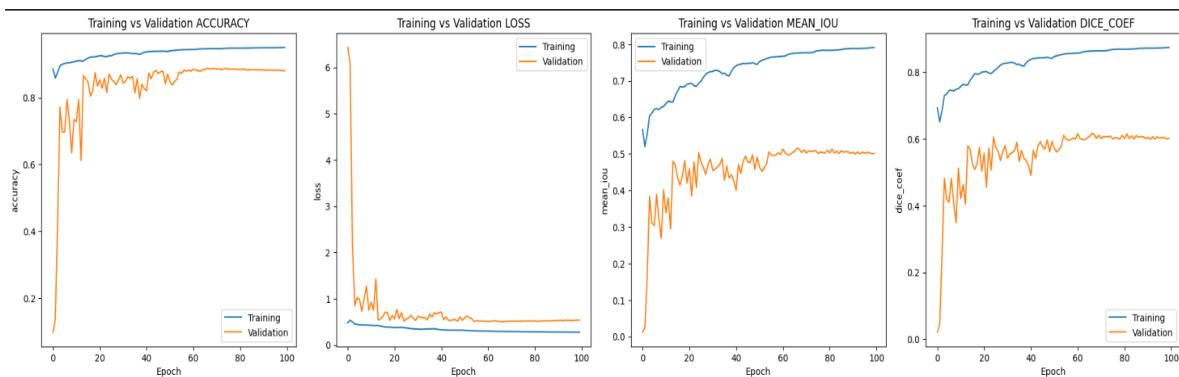
همچنین epoch و batch\_size=32 را نیز برابر با 100 قرار می‌دهیم چرا که در مقاله ذکر شده تعداد epoch بالا در کارایی این مدل تاثیر مثبت خواهد داشت.

در نهایت پس از پایان 100 epoch به پارامترهای زیر رسیدیم:

accuracy: 0.9462 - dice\_coef: 0.8685 - loss: 0.2809 - mean\_iou: 0.7862 - val\_accuracy: 0.8805 - val\_dice\_coef: 0.6013 - val\_loss: 0.5380 - val\_mean\_iou: 0.5008

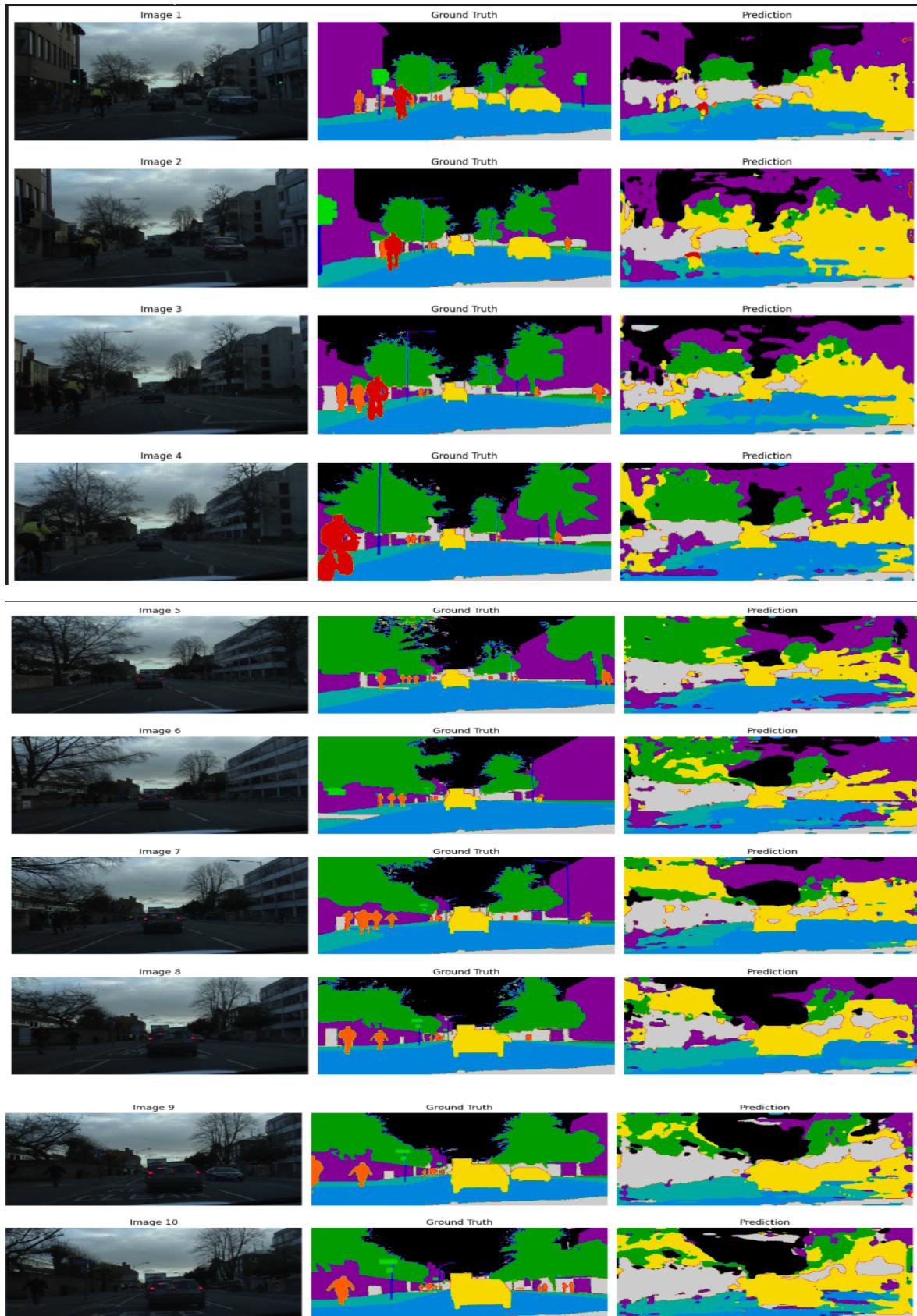
بنابراین طبق گفته صورت پژوه به خواسته‌های بیان شده رسیده‌ایم و  $mIoU$  و dice coefficient بالای 0.5 روی داده‌های اعتبارسنجی داریم.

نمودارهای خواسته شده در هنگام آموزش در زیر آمده‌است:



شکل 11. نمودارهای خواسته شده در فرایند آموزش

## 6-1. ارزیابی مدل



شکل 12. تصاویر مربوط به داده‌های تست به همراه ماسک اصلی و پیش‌بینی شده

با توجه به مقدار mIoU که در اینجا 0.5 است نتایج بدست آمده منطقی است چرا که مقدار 0.5 در این جا به این معنی است که بطور میانگین 50٪ از پیکسل های واقعی هر کلاس با پیش بینی هم پوشانی دارند. کلاس هایی که پیکس زیادی در تصویر دارند مانند جاده یا آسمان تاثیر بیشتری روی mIoU دارند و اگر آنها درست پیش بینی شوند باز هم مقدار mIoU خوبی خواهیم داشت. در اینجا نیز مشاهده می شود که کلاس های بزرگتر مانند جاده و آسمان با دقت بالا تشخیص داده شده اند. همچنین شکل کلی کلاس های متوسط مثل ماشین یا ساختمان به درستی شناسایی شده و ساختار کلی صحنه مثل افق و مسیر خیابان درک شده اما اشیای کوچک یا باریک شاید خیلی خوب تشخیص داده نشوند و مرز بین کلاس ها نیز چندان دقیق نیست و کلاس های مشابه مانند پیاده رو و جاده نیز گاهها اشتباه تشخیص داده می شوند. برای بهبود عملکرد در اینجا طبق گفته مقاله data augmentation به میزان بالا تاثیر مثبتی خواهد داشت و با توجه به کوچک بودن دیتا است CamVid احتمالا اعمال این روش دقت را بالاتر خواهد برد. همچنین اگر از تصاویر با رزولوشن بالاتر استفاده کنیم با حفظ جزئیات تصویر کلاس های کوچکتر بهتر تشخیص داده خواهند شد.

## پرسش ۲. Oriented R-CNN برای تشخیص اشیا

بخش اول:

الف.

انگیزه اصلی توسعه Oriented R-CNN و رفع محدودیت‌های روش‌های قبلی انگیزه اصلی توسعه مدل Oriented R-CNN از نیاز به تشخیص دقیق اجسام با جهت‌گیری‌های مختلف در تصاویر نشأت می‌گیرد، بهویژه در حوزه‌هایی مانند تصاویر ماهواره‌ای و هوایی که اجسام معمولاً به صورت افقی یا عمودی قرار نگرفته‌اند. روش‌های کلاسیک مانند Faster R-CNN و سایر مدل‌های مبتنی بر جعبه‌بندی افقی (Horizontal Bounding Box)، به دلیل اینکه فقط جعبه‌های مستطیلی محوردار تولید می‌کردند، نمی‌توانستند اجسام با زاویه و جهت‌گیری دلخواه را به خوبی تشخیص دهند؛ این موضوع باعث می‌شد بخش زیادی از پس‌زمینه یا حتی اشیاء دیگر درون جعبه انتخاب شود و دقت مدل کاهش یابد. در روش‌های قبلی تولید پروپوزال (پیشنهاد جعبه)، معمولاً از دو راه حل عمدۀ استفاده می‌شد:

Rotated RPN : این روش با قراردادن تعداد زیادی انکر با زوایا، مقیاس‌ها و نسبت‌های مختلف (مثلاً ۵۴ انکر در هر موقعیت) تلاش می‌کرد همه حالات چرخش را پوشش دهد. این کار باعث افزایش یادآوری (recall) مدل می‌شد اما هزینه محاسباتی و حافظه بسیار بالا می‌رفت و در تصاویر با اجسام متراکم یا اندازه‌های مختلف عملکرد مناسبی نداشت.

RoI Transformer و RoI Alignment : این مدل ابتدا جعبه‌های افقی را تولید و سپس با عملیات پیچیده‌ای مثل کاهش تعداد انکرها می‌شد، اما همچنان محاسبات سنگین و زمان‌بر بود و شبکه سنگینی را ایجاد می‌کرد.

محدودیت‌های اصلی روش‌های قبلی:

حجم محاسبات بسیار زیاد در مرحله تولید پروپوزال‌ها به دلیل استفاده از انکرها متعدد با زوایا و نسبت‌های مختلف.

صرف حافظه بالا و کاهش سرعت inference (استنتاج) مدل.

نیاز به عملیات پیچیده برای تبدیل جعبه‌های افقی به چرخیده.

ریسک بالای overfitting به دلیل تعداد بالای پارامترها.

راه حل Oriented R-CNN: Oriented R-CNN با معرفی یک Oriented RPN جدید و ساده، به شکل مستقیم و تقریباً بدون هزینه اضافی (cost-free)، جعبه‌های پیشنهادی چرخیده (Oriented Proposals) را تولید می‌کند. در این روش، بجای استفاده از تعداد زیاد انکر و عملیات پیچیده، از یک روش نمایشی جدید به نام midpoint offset representation توصیف می‌شود که امکان تخمین موقعیت در این روش، هر جعبه با شش پارامتر ( $x, y, w, h, \Delta\alpha, \Delta\beta$ ) توصیف می‌شود که امکان تخمین موقعیت و چرخش جسم را به سادگی و با دقت بالا فراهم می‌کند.

مزیت‌های Oriented R-CNN نسبت به روش‌های قبل:

تعداد پارامترهای Oriented RPN نسبت به ROI Transformer تقریباً  $3000/1$  و نسبت به Rotated RPN تقریباً  $15/1$  است، بنابراین بسیار سبک‌تر و سریع‌تر است.

در عین سادگی، دقت مدل روی دیتاست‌های معتر (مانند DOTA و HRSC $2016$ ) به شکل قابل توجهی از روش‌های قبلی بیشتر است. به دلیل ساختار سبک‌تر، ریسک overfitting کاهش یافته و سرعت اجرای مدل تقریباً مشابه مدل‌های تک مرحله‌ای شده است. این مدل بدون نیاز به ترفندهای اضافی و فقط با استفاده از شبکه‌های backbone معمول (مثل ResNet $50$ ) بهترین نتایج را ثبت کرده است. مثال: در Rotated RPN اگر قرار باشد برای هر مکان در تصویر  $54$  انکر قرار گیرد، برای تصویری با ابعاد بزرگ هزاران انکر تولید می‌شود که بار محاسباتی و حافظه را شدیداً بالا می‌برد. اما در Oriented R-CNN، با همان دقت یا حتی بیشتر، با پارامترهای کمتر و ساختار سبک‌تر می‌توان جعبه‌های چرخیده را تولید کرد. همچنین، برخلاف ROI Transformer که نیازمند چندین لایه fully connected عملیات سنگین است، Oriented R-CNN این فرایند را به صورت end-to-end و سریع انجام می‌دهد.

(ب)

مزایای استفاده از نمایش mid-point offset نسبت به نمایش‌های سنتی جعبه‌های محدود‌کننده (Bounding Box):

نمایش سنتی جعبه‌های محدود‌کننده، معمولاً به صورت مستطیل افقی (axis-aligned bounding box) تعریف می‌شود که با چهار پارامتر ( $x, y, w, h$ ) (مرکز جعبه، عرض و ارتفاع) یا گوش‌های بالا-چپ و پایین-راست مشخص می‌گردد.

این روش برای اشیایی که به طور افقی یا عمودی در تصویر قرار دارند مناسب است، اما وقتی شیء دارای زاویه یا چرخش است، این جعبه نمی‌تواند موقعیت آن را بدقت بیان کند و معمولاً بخش زیادی از پس زمینه یا اشیاء دیگر را نیز شامل می‌شود. نمایش mid-point offset با افروden دو پارامتر اضافی ( $\Delta\alpha$  و  $\Delta\beta$ ) علاوه بر پارامترهای معمول، موقعیت جعبه را با توجه به زاویه و چرخش جسم توصیف می‌کند.

در این روش: ( $X, Y$ ): مرکز جعبه محدود‌کننده افقی

(external rectangle): عرض و ارتفاع جعبه محدود‌کننده افقی ( $w, h$ )

( $\Delta\alpha, \Delta\beta$ ): آفست (جابجایی) میانه ضلع بالا و ضلع راست نسبت به نقاط متناظر جعبه افقی، که تعیین‌کننده جهت‌گیری جعبه چرخیده است.

مزایای اصلی: افزایش دقت مکان‌یابی برای اجسام چرخیده با استفاده از پارامترهای آفست، مدل می‌تواند جعبه‌ای دقیقاً منطبق با مرزهای جسم (صرف نظر از زاویه قرار گیری) رسم کند. این باعث کاهش فضای اضافی (پس زمینه) و افزایش دقت در بررسی اجسام می‌شود.

مثال: فرض کنید یک کشتی یا هواپیما به صورت مورب در تصویر قرار دارد؛ جعبه افقی معمولی مقدار زیادی پس زمینه را نیز شامل می‌شود، اما با mid-point offset جعبه دقیقاً روی لبه‌های شیء قرار می‌گیرد. کاهش پیچیدگی و پارامترهای مدل نمایش‌های کلاسیک برای نمایش جعبه‌های چرخیده باید

تعداد زیادی انکر با زوایا و مقیاس‌های مختلف تولید کنند، در حالی که midpoint offset تنها با شش پارامتر، همان هدف را محقق می‌کند. این کار باعث ساده‌تر شدن شبکه و کاهش احتمال overfitting می‌شود.

مثال: در روش Rotated RPN، به ازای هر موقعیت باید ۵۴ انکر تعریف شود؛ ولی با midpoint offset، تنها سه انکر کافی است.

یکپارچگی با مکانیزم رگرسیون افقی

در این روش می‌توان همزمان از مزایای رگرسیون جعبه افقی (که اثبات شده و پایدار است) و نمایش چرخیده بهره برد.

مدل به جای یادگیری ساختار کاملاً جدید، تنها دو پارامتر آفست اضافه یاد می‌گیرد.

مثال: شبکه‌های مبتنی بر Faster R-CNN را می‌توان به سادگی با تغییر شاخه رگرسیون به نسخه چرخیده بهینه کرد، بدون نیاز به بازطراحی کل ساختار.  
تفسیر و بازسازی ساده جعبه‌های چرخیده

محاسبه رأس‌های جعبه با فرمول‌های ساده جبری و فقط با همین شش پارامتر انجام می‌شود، در حالی که مدل‌های دیگر ممکن است نیازمند محاسبات پیچیده‌تری باشند.

مثال: با داشتن  $(x, y, w, h, \Delta\alpha, \Delta\beta)$ ، می‌توان چهار رأس جعبه را بلاfacله محاسبه و روی تصویر نمایش داد.

جمع‌بندی با یک مثال ساده: فرض کنید بخواهیم یک کانتینر کشته که در زاویه  $30^\circ$  درجه قرار گرفته را در یک تصویر ماهواره‌ای تشخیص دهیم: روش افقی: جعبه‌ای مستطیلی که مقدار زیادی از فضای کناری آب یا کشتی‌های مجاور را نیز در بر می‌گیرد و مرزهای جسم را دقیق نمی‌پوشاند. midpoint offset جعبه‌ای چرخیده با شش پارامتر که دقیقاً روی لبه‌های کانتینر قرار می‌گیرد، هیچ جسم اضافه یا پس‌زمینه‌ای را شامل نمی‌شود و محل، زاویه و بعد کانتینر را به دقت بیان می‌کند.

اجزای مدل :

(الف )

## معماری Oriented RPN و تفاوت آن با RPN سنتی

۱. معماری Oriented RPN یک بخش کلیدی در معماری Oriented RPN است که وظیفه تولید جعبه‌های پیشنهادی (proposals) با قابلیت چرخش (زاویه‌دار) را دارد. معماری آن شامل موارد زیر است:

ورویدی: پنج سطح ویژگی (feature maps) از شبکه Oriented RPN به FPN داده می‌شود:  
 $\{P_2, P_3, P_4, P_5, P_6\}$

ساختار شبکه: در هر سطح، یک لایه کانولوشن  $3 \times 3$  و دو لایه کانولوشن  $1 \times 1$  موازی قرار داده می‌شود. یکی از این لایه‌های  $1 \times 1$  برای رگرسیون (یادگیری پارامترهای جعبه پیشنهادی) و دیگری برای پیش‌بینی احتمال وجود شیء (objectness score) است.

انکرها (Anchors): برخلاف روش‌های چرخیده کلاسیک، Oriented RPN فقط سه انکر افقی با نسبت‌های مختلف (۱:۲، ۱:۱، ۲:۱) در هر موقعیت قرار می‌دهد (یعنی نیازی به انکرهای متعدد با زوایای مختلف نیست). ابعاد انکرها با توجه به سطح ویژگی تعیین می‌شود (مثلاً  $32 \times 32$  تا  $512 \times 512$  پیکسل).

پارامترهای رگرسیون:

به جای چهار پارامتر Oriented RPN شش پارامتر خروجی می‌دهد:  $(x, y, w, h, \Delta\alpha, \Delta\beta)$

این پارامترها مرکز، عرض، ارتفاع و آفست‌های میانه ضلع بالا و راست جعبه را مشخص می‌کنند که در نهایت جعبه‌های چرخیده دقیق تولید می‌شود.

خروجی: برای هر انکر، شش مقدار (رگرسیون) + مقدار احتمال (کلاس) جعبه‌های پیشنهادی چرخیده نهایی با همین اطلاعات ساخته می‌شوند.

### : Rotated RoIAlign

(الف)

هدف اصلی Rotated RoIAlign این است که ویژگی‌های دقیق و باکیفیت را از مناطق پیشنهادی چرخیده (Oriented Proposals) استخراج کند. این کار مخصوصاً زمانی اهمیت پیدا می‌کند که اجسام در تصویر با زاویه و جهت دلخواه قرار گرفته باشند و دیگر جعبه‌های افقی معمول نتوانند ویژگی‌های خوبی برای شبکه فراهم کنند. Rotated RoIAlign کمک می‌کند ویژگی‌های استخراج شده دقیقاً مطابق با زاویه و موقعیت واقعی جسم باشد، در نتیجه طبقه‌بندی و مکان‌یابی دقیق‌تر انجام می‌شود.

#### مراحل انجام عملیات Rotated RoIAlign

۱. دریافت جعبه پیشنهادی چرخیده: ابتدا مدل، یک جعبه پیشنهادی با چهار رأس (که معمولاً به صورت متوازی‌الاضلاع است) به عنوان منطقه مورد بررسی انتخاب می‌کند.

۲. تبدیل به مستطیل چرخیده: این جعبه متوازی‌الاضلاع برای ساده‌تر شدن محاسبات به یک مستطیل چرخیده (oriented rectangle) تبدیل می‌شود. برای این کار، طول قطر کوتاه‌تر با قطر بلندتر برابر می‌شود تا ساختار مستطیل حفظ شود. پارامترهای این مستطیل شامل مرکز ( $x, y$ )، عرض ( $w$ )، ارتفاع ( $h$ ) و زاویه چرخش ( $\theta$ ) هستند.

۳. نگاشت (Project) جعبه روی نقشه ویژگی (Feature Map): ابعاد و مختصات جعبه چرخیده با توجه به مقیاس (stride) شبکه روی feature map اعمال می‌شود. به این ترتیب، مختصات جعبه روی نقشه ویژگی تنظیم می‌شود.

۴. تقسیم جعبه به شبکه کوچک‌تر: مستطیل چرخیده به یک شبکه کوچک‌تر (مثلاً  $7 \times 7$  سلوول) تقسیم می‌شود. این شبکه باعث می‌شود بتوان از هر قسمت جعبه، ویژگی جداگانه‌ای استخراج کرد.

۵. نمونه‌برداری (Sampling) چرخشی از هر سلوول: برای هر سلوول از این شبکه، با توجه به زاویه چرخش جعبه، مختصات سلوول محاسبه می‌شود و نمونه‌برداری از پیکسل‌های از همان سلوول به همان بخش انجام می‌شود. به این ترتیب، ویژگی هر سلوول با ساختار چرخیده جعبه هم‌راستا خواهد بود.

۶. تولید بردار ویژگی نهايی: در پايان، ویژگی‌های استخراج شده از تمام سلول‌ها در قالب يک بردار ویژگی با اندازه ثابت (مثلاً  $7 \times 7 \times C$ ) ذخیره می‌شود و برای طبقه‌بندی يا رگرسيون نهايی به لاييه‌های بعدی شبکه داده می‌شود.

مثلاً می‌خواهيم يك کشتی را که با زاویه ۴۵ درجه در تصویر قرار گرفته تشخيص دهيم. اگر از **RoIAlign** معمولی استفاده شود، ویژگی‌های مربوط به کشتی به صورت افقی استخراج می‌شوند و بخش‌هایی از کشتی در سلول‌های نامربوط قرار می‌گيرند و برخی اطلاعات دقیق جسم از دست می‌رود. اما با **Rotated RoIAlign**، جعبه پیشنهادی با زاویه کشتی تراز می‌شود و سلول‌های شبکه ویژگی دقیقاً مطابق با ساختار و جهت واقعی کشتی تشکیل می‌شوند. به این ترتیب، اطلاعات کامل تر و دقیق‌تری از کشتی استخراج می‌شود و مدل با اطمینان بیشتری می‌تواند آن را شناسایی و موقعیت‌یابی کند.

(ب)

اگر در مدل‌های تشخيص اجسام چرخیده به جای **Rotated RoIAlign** از روش‌های سنتی مانند **RoIAlign** افقی استفاده شود، مشکلات زیر به وجود خواهد آمد:

۱. ناهماهنگی ویژگی‌ها با ساختار و زاویه واقعی جسم وقتی اجسام با زاویه در تصویر قرار دارند، سلول‌های ویژگی افقی با مرز واقعی جسم منطبق نیستند و ویژگی‌های استخراج شده ترکیبی از جسم و پس‌زمینه يا حتی اجسام دیگر خواهد بود. مثال: اگر يك کشتی با زاویه ۵۰ درجه در تصویر ماهواره‌ای باشد و استخراج ویژگی به صورت افقی انجام شود، اطلاعات آب اطراف و اجسام دیگر با ویژگی‌های کشتی مخلوط می‌شود و مدل نمی‌تواند کشتی را به درستی تشخيص دهد.

۲. کاهش دقت در طبقه‌بندی و مکان‌یابی ویژگی‌های نامتناسب باعث می‌شود مدل در تشخيص کلاس جسم و تعیین موقعیت و زاویه صحیح آن دچار خطا شود. این مشکل به ویژه برای اجسام کوچک يا با جهت‌های خاص بیشتر دیده می‌شود. مثال: در تشخيص متون يا پلاک‌های چرخیده، اگر ویژگی‌ها افقی استخراج شوند، مدل ممکن است بخشنی از متن يا پلاک را به درستی شناسایی نکند يا جهت آن را اشتباه پیش‌بینی کند.

۳. افزایش تداخل و ادغام ویژگی‌های اجسام مجاور در تصاویری با اجسام پرتراکم و با زاویه‌های مختلف، اگر استخراج ویژگی فقط افقی باشد، ویژگی اجسام مجاور با هم ترکیب می‌شود و مدل نمی‌تواند هر جسم را به درستی از بقیه جدا کند. مثال: در پارکینگی با خودروهایی که هر کدام با زاویه متفاوت پارک شده‌اند، ویژگی دو خودرو نزدیک به هم ممکن است با هم ترکیب شود و مدل اشتباه کند.

۴. از دست رفتن اطلاعات زاویه‌ای و ساختاری روش‌های سنتی فقط مختصات افقی و عمودی را پوشش می‌دهند و هیچ اطلاعاتی درباره زاویه و چرخش جسم ارائه نمی‌کنند. این موضوع به ویژه در کاربردهای مانند تصاویر هوایی، صنعتی يا تشخیص متن و پلاک چرخیده، باعث افت دقت مدل می‌شود.

عملکرد و کارایی :

(الف)

## ۱. نوآوری‌های معماری Oriented R-CNN

الف) طراحی Oriented RPN سبک و مؤثر در اولین مرحله، شبکه‌ی پیشنهاد منطقه (Oriented RPN) به صورت بسیار سبک و کارآمد طراحی شده است. این شبکه با استفاده از روش جدید «نمایش

آفست نقطه میانی» (midpoint offset representation)، به جای استفاده از تعداد زیادی anchor با زوایای مختلف، هر جعبه مایل را تنها با شش پارامتر توصیف می‌کند. این رویکرد باعث کاهش قابل توجه پیچیدگی و افزایش سرعت مدل می‌شود و در عین حال، دقت بالای پیشنهادها را حفظ می‌کند.

ب) استفاده از Rotated RoIAlign برای هر Oriented R-CNN Head در مرحله دوم، proposal مايل، ويژگى ها را با عمليات Rotated RoIAlign استخراج می کند. اين کار باعث می شود ويژگى های استخراج شده با جهت واقعی شیء کاملاً منطبق باشند و مشکل عدم تطابق ويژگى و شیء که در بسیاری از روش های دیگر دیده می شود، برطرف شود. ج) سادگی و پارامترهای کمتر تعداد پارامترهای rotated RPN حدود ۱۵/۱ +RoI Transformer نسخه ۳۰۰۰/۱ Oriented RPN است. این کاهش پارامترها علاوه بر افزایش سرعت، ریسک overfitting را نیز کاهش می دهد.

#### شواهد تجربی (آزمایش‌ها و نتایج مقاله)

الف) نتایج روی دیتابست DOTA بر اساس جدول ۲ مقاله، backbone با Oriented R-CNN از نوع ResNet-۵۰-FPN موفق به کسب mAP برابر با ۷۵.۸۷٪ شده است که نسبت به همه روش‌های پیشین با همان backbone بالاتر است. همچنین در حالت multi-scale این عدد به ۸۰.۸۷٪ می‌رسد که جزو بهترین نتایج است.

ب) نتایج روی دیتابست HRSC۲۰۱۶ در جدول ۳ مقاله، backbone با Oriented R-CNN از نوع ResNet-۵۰-FPN به mAP معادل ۹۶.۵۰٪ رسیده که بالاتر از تمام مدل‌های مقایسه شده است. ج) سرعت و کارایی بالا در جدول ۴ مقاله، سرعت اجرای مدل روی تصویر  $1024 \times 1024$  برابر ۱۵.۱ FPS گزارش شده است که تقریباً معادل مدل‌های یک مرحله‌ای است اما دقت آن به مراتب بالاتر است.

به لطف معماری ساده، استفاده از Oriented RPN سبک و عمليات Rotated RoIAlign، دقت و کارایی بالاي را به طور همزمان ارائه می دهد. نتایج آزمایش‌ها نشان می دهند که این مدل نه تنها از نظر دقت (mAP) بلکه از نظر سرعت اجرا نیز بر بسیاری از مدل‌های پیشین برتری دارد. ارجاع به مقاله: این نتایج به طور مشخص در جدول‌های ۲، ۳ و ۴ و نیز بخش‌های ۴.۳ تا ۴.۵ مقاله آمده است.

(ب)

عوامل مؤثر در کارایی محاسباتی چارچوب Oriented R-CNN و نقش هر عامل  
۱. Oriented RPN سبک و کم‌پارامتر در مرحله اول، Oriented RPN برای تولید proposal‌های زاویه‌دار به کار می‌رود. این شبکه به جای استفاده از تعداد زیادی anchor با زوایای مختلف، تنها با سه anchor افقی با نسبت‌های متفاوت و شش پارامتر برای هر جعبه عمل می‌کند.  
نقش: این ساختار باعث کاهش شدید تعداد پارامترهای کاهش مصرف حافظه و پایین آمدن زمان محاسباتی می‌شود. در نتیجه تولید proposal‌ها سریع و تقریباً بدون هزینه انجام می‌شود.

۲. نمایه‌سازی midpoint offset representation در این روش، هر جعبه زاویه‌دار تنها با شش پارامتر ( $X, y, w, h, \Delta\alpha, \Delta\beta$ ) توصیف می‌شود و نیازی به مختصات پیچیده یا چهار گوشه نیست. نقش: این سادگی باعث می‌شود فرآیند یادگیری مدل پایدارتر، سریع‌تر و با وابستگی کمتر به عملیات محاسباتی اضافه صورت گیرد.

۳. عملیات برای استخراج ویژگی‌های جعبه‌های زاویه‌دار، از عملیات Rotated RoIAlign استفاده می‌شود که ویژگی‌ها را با دقت و بدون حساسیت به زاویه استخراج می‌کند. نقش: این عملیات مانع بروز عدم تطابق بین ویژگی‌ها و شیء اصلی می‌شود و باعث افزایش دقت نهایی مدل با حداقل محاسبات اضافه می‌گردد.

۴. استفاده از Non-Maximum Suppression (NMS) بهینه در مرحله تست، ابتدا از horizontal NMS برای حذف proposal‌های تکراری و سپس از ploy NMS روی خروجی نهایی استفاده می‌شود. نقش: این کار باعث می‌شود فقط proposal‌های مهم و غیرتکراری به مرحله بعد بررسند و سرعت پردازش مدل بالا بماند.

۵. کاهش حجم محاسبات و حافظه در کل معماری، به جای استفاده از fully connected سنگین، از شبکه‌های convolutional سبک استفاده شده است. نقش: این انتخاب هم در آموزش و هم در اجرا باعث مصرف کمتر منابع و اجرای سریع‌تر مدل می‌شود.

۶. ساختار دو مرحله‌ای بهینه‌شده با ساده‌سازی اجزای مدل، Oriented R-CNN علی‌رغم دو مرحله‌ای بودن، تقریباً با سرعت مدل‌های یک مرحله‌ای عمل می‌کند، اما دقت بسیار بالاتری ارائه می‌دهد. نقش: این ساختار سبب می‌شود که مدل در شرایط عملی و واقعی کاملاً کاربردی باشد و نیاز به منابع زیاد نداشته باشد.

(ج)

تحلیل انتقادی کوتاه مقایسه Oriented R-CNN با سایر آشکارسازهای جهت‌دار دو مرحله‌ای نقاط قوت Oriented R-CNN

سادگی و سرعت بالا: Oriented R-CNN نسبت به مدل‌های دو مرحله‌ای مانند ROI Transformer و SCRDet ساختاری بسیار ساده‌تر دارد و با استفاده از Oriented RPN سبک و کم‌پارامتر و حذف anchor‌های چرخیده متعدد، سرعت اجرا را به طور چشمگیری افزایش داده است. این موضوع باعث شده است که مدل حتی در سخت‌افزارهای معمولی نیز به راحتی اجرا شود.

دقت بالاتر: بر اساس نتایج مقاله، Oriented R-CNN موفق شده است mAP بالاتری نسبت به سایر مدل‌های دو مرحله‌ای کسب کند (مثلاً ۷۵.۸۷٪ روی دیتاست DOTA و ۹۶.۵۰٪ روی HRSC2016).

حتی با backbone سبک‌تر مثل ResNet-50-FPN، عملکرد این مدل از بسیاری مدل‌های سنگین‌تر بهتر بوده است. پیاده‌سازی و تنظیم آسان‌تر: به دلیل سادگی معماری و حذف تنظیمات anchor پیچیده، مدل Oriented R-CNN برای پیاده‌سازی و توسعه بسیار راحت‌تر است و نیاز به آزمون و خطای فراوان در تنظیمات ندارد.

## نقاط ضعف Oriented R-CNN

عدم استفاده از بعضی ویژگی‌های پیشرفته سایر مدل‌ها: برخی مدل‌های پیشرفته‌تر مثل SCRDet یا ROI Transformer از تکنیک‌های پیشرفته‌تر برای alignment و استفاده از context بهره می‌برند که در تشخیص اشیاء کوچک یا بسیار پیچیده می‌تواند مزیت داشته باشد.

وابستگی به Rotated RoIAlign دقت استخراج ویژگی‌ها را بالا می‌برد، اما یک مرحله اضافی است که ممکن است در سخت‌افزارهای بسیار محدود چالش ایجاد کند و همچنان محاسبات بیشتری نسبت به مدل‌های یک مرحله‌ای نیاز دارد.

تمرکز بر تصاویر خاص: بیشترین آزمایش‌های مدل بر روی تصاویر هوایی و اشیای دارای زاویه بوده است. بنابراین ممکن است برای دیتاست‌های متفاوت یا کاربردهای غیرهوافضایی نیاز به تنظیمات مجدد یا بررسی بیشتر باشد.

## بخش دوم : پیاده سازی عملی راه اندازی محیط و آماده سازی مجموعه داده

```
import os
import xml.etree.ElementTree as ET
import numpy as np
import torch
from torch.utils.data import Dataset
import cv2
from torchvision import transforms
import matplotlib.pyplot as plt
import matplotlib.patches as patches

def read_image_list(dataset_path, split='train'):
    split_file = os.path.join(dataset_path, 'ImageSets',
f'{split}.txt')
    with open(split_file, 'r') as f:
        image_ids = [line.strip() for line in f if line.strip()]
    return image_ids

def parse_xml_annotation(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    objects = []
    for obj in root.findall('object'):
        robndbox = obj.find('robndbox')
        cx = float(robndbox.find('cx').text)
        cy = float(robndbox.find('cy').text)
        w = float(robndbox.find('w').text)
        h = float(robndbox.find('h').text)
        angle = float(robndbox.find('angle').text)
        label = obj.find('name').text
```

```

        objects.append({
            'robndbox': {'cx': cx, 'cy': cy, 'w': w, 'h': h, 'angle': angle},
            'label': label
        })
    return objects

def robndbox_to_vertices(cx, cy, w, h, angle):
    angle = -angle
    cos_a = np.cos(angle)
    sin_a = np.sin(angle)
    hw, hh = w / 2, h / 2
    vertices = [(-hw, -hh), (hw, -hh), (hw, hh), (-hw, hh)]
    rotated_vertices = []
    for x, y in vertices:
        x_rot = x * cos_a + y * sin_a
        y_rot = -x * sin_a + y * cos_a
        rotated_vertices.append((cx + x_rot, cy + y_rot))
    return rotated_vertices

def vertices_to_midpoint_offset(vertices):
    xs = [v[0] for v in vertices]
    ys = [v[1] for v in vertices]
    x = np.mean(xs)
    y = np.mean(ys)
    w = max(xs) - min(xs)
    h = max(ys) - min(ys)
    return {'x': x, 'y': y, 'w': w, 'h': h}

def preprocess_dataset(dataset_path, split='train'):
    image_ids = read_image_list(dataset_path, split)
    data = []
    image_dir = os.path.join(dataset_path, 'AllImages')
    annotation_dir = os.path.join(dataset_path, 'Annotations')
    for image_id in image_ids:
        image_path = os.path.join(image_dir, f'{image_id}.bmp')
        xml_path = os.path.join(annotation_dir, f'{image_id}.xml')
        if os.path.exists(image_path) and os.path.exists(xml_path):
            objects = parse_xml_annotation(xml_path)
            annotations = []
            for obj in objects:
                robndbox = obj['robndbox']
                vertices = robndbox_to_vertices(
                    robndbox['cx'], robndbox['cy'], robndbox['w'],
                    robndbox['h'], robndbox['angle'])
                )
                midpoint_offset =
vertices_to_midpoint_offset(vertices)

```

```

        # Store both midpoint-offset and original robndbox
        annotations.append({
            'midpoint': midpoint_offset,
            'robndbox': robndbox
        })
    labels = [obj['label'] for obj in objects]
    #from XML
    tree = ET.parse(xml_path)
    root = tree.getroot()
    size = root.find('size')
    orig_width = int(size.find('width').text)
    orig_height = int(size.find('height').text)
    data.append({
        'image_path': image_path,
        'annotations': annotations,
        'labels': labels,
        'orig_size': (orig_width, orig_height)
    })
return data

class HRSC2016Dataset(Dataset):
    def __init__(self, dataset_path, split='train', transform=None):
        self.data = preprocess_dataset(dataset_path, split)
        self.transform = transform
        self.label_to_idx = {'ship': 0}

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image_path = self.data[idx]['image_path']
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        annotations = self.data[idx]['annotations']
        labels = [self.label_to_idx[label] for label in
        self.data[idx]['labels']]
        #Prepare midpoint-offset boxes
        midpoint_boxes = [
            [ann['midpoint']['x'], ann['midpoint']['y'],
            ann['midpoint']['w'], ann['midpoint']['h']]
            for ann in annotations
        ]
        robndboxes = [ann['robndbox'] for ann in annotations]
        midpoint_boxes = torch.tensor(midpoint_boxes,
        dtype=torch.float32) if midpoint_boxes else torch.empty((0, 4),
        dtype=torch.float32)
        labels = torch.tensor(labels, dtype=torch.long) if labels
        else torch.empty((0,), dtype=torch.long)

```

```

        orig_size = self.data[idx]['orig_size']
        if self.transform:
            image = self.transform(image)
        return image, {
            'midpoint_boxes': midpoint_boxes,
            'robndboxes': robndboxes,
            'labels': labels,
            'image_id': os.path.basename(image_path).replace('.bmp',
            ''),
            'orig_size': orig_size
        }

def get_dataset(dataset_path, split='train'):
    transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((512, 512)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
    ])
    return HRSC2016Dataset(dataset_path, split, transform)

def visualize_sample(image, midpoint_boxes, robndboxes, labels,
image_id, orig_size):
    image = image.permute(1, 2, 0).numpy()
    std = np.array([0.229, 0.224, 0.225])
    mean = np.array([0.485, 0.456, 0.406])
    image = (image * std + mean).clip(0, 1)

    orig_width, orig_height = orig_size
    scale_x = 512 / orig_width
    scale_y = 512 / orig_height

    fig, ax = plt.subplots()
    ax.imshow(image)

    #rotated bounding boxes(red)
    for robndbox in robndboxes:
        vertices = robndbox_to_vertices(
            robndbox['cx'] * scale_x,
            robndbox['cy'] * scale_y,
            robndbox['w'] * scale_x,
            robndbox['h'] * scale_y,
            robndbox['angle'])
        )
        xs, ys = zip(*vertices + vertices[:1])

```

```

        ax.plot(xs, ys, 'r-', linewidth=2, label='Rotated Box' if
robndbox == robndboxes[0] else "")

#midpoint-offset boxes (blue)
for box in midpoint_boxes:
    x, y, w, h = box.numpy()
    x, y, w, h = x * scale_x, y * scale_y, w * scale_x, h *
scale_y
    rect = patches.Rectangle(
        (x - w/2, y - h/2), w, h,
        linewidth=2, edgecolor='b', facecolor='none',
        label='Axis-Aligned Box' if box is midpoint_boxes[0] else
"""
)
ax.add_patch(rect)

plt.title(f"Image ID: {image_id}, Labels: {labels.tolist()}")
plt.legend()
plt.axis('off')
plt.show()

if __name__ == "__main__":
    dataset_path = '/kaggle/input/hrsc2016-ms-dataset'
    dataset = get_dataset(dataset_path, split='train')
    for i in range(9): # Visualize only three samples
        image, target = dataset[i]
        visualize_sample(
            image,
            target['midpoint_boxes'],
            target['robndboxes'],
            target['labels'],
            target['image_id'],
            target['orig_size']
        )

```

Image ID: 100000883, Labels: [0, 0, 0, 0, 0, 0]

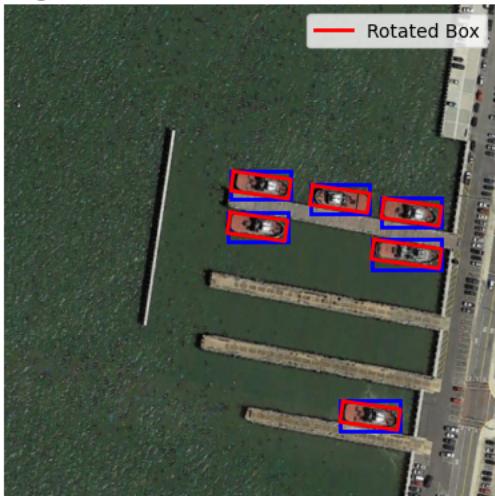


Image ID: 100001119, Labels: [0, 0, 0, 0, 0]

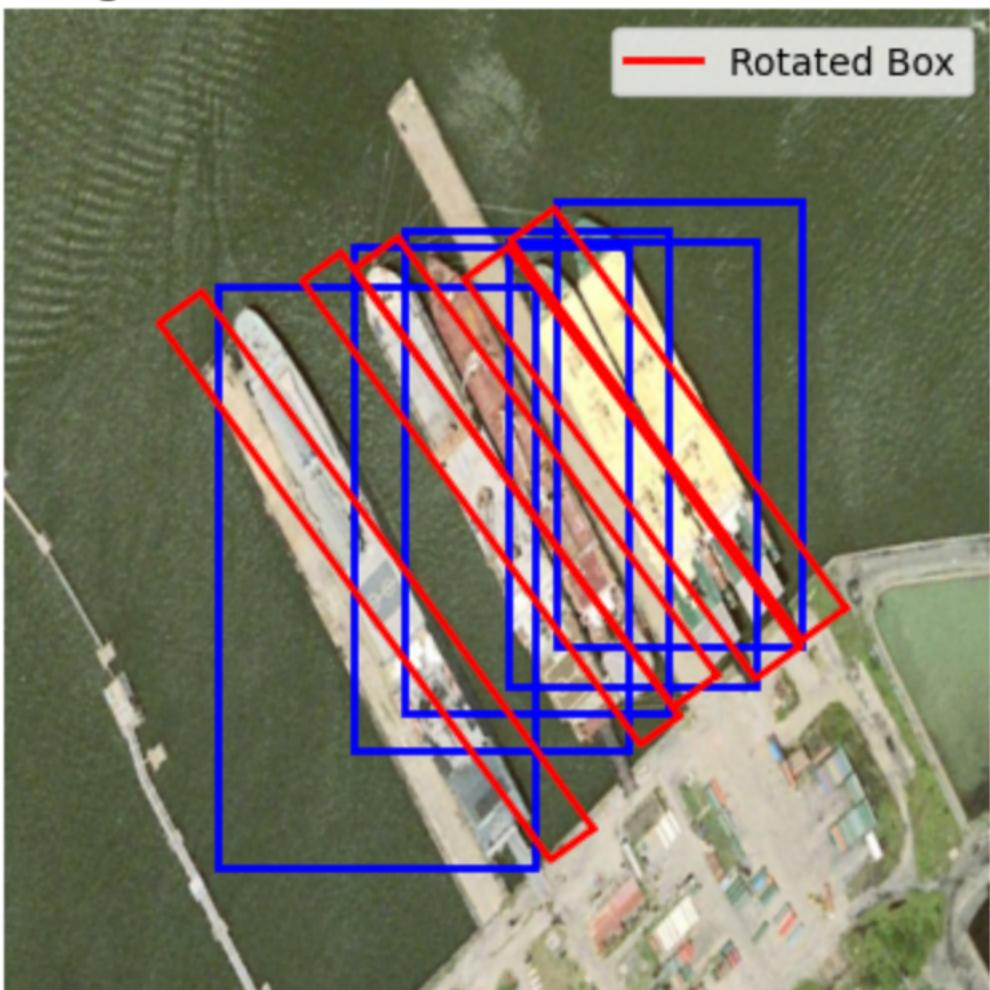
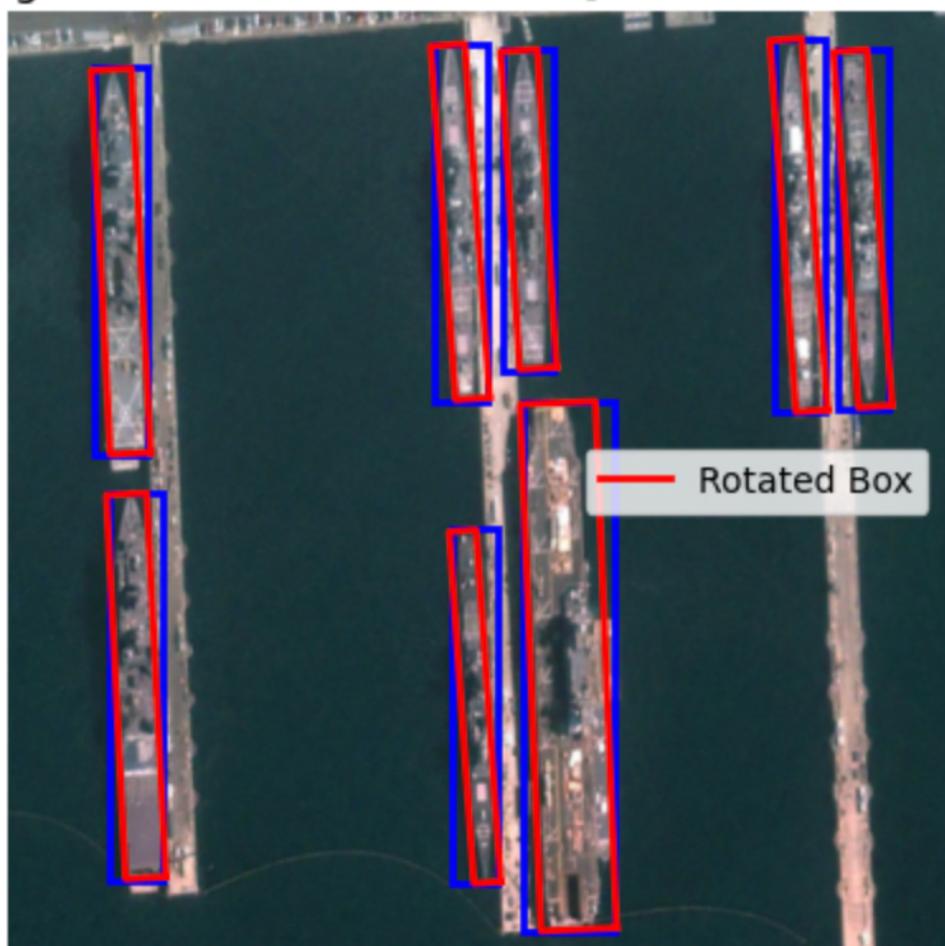


Image ID: 100001554, Labels: [0, 0, 0, 0, 0, 0, 0, 0, 0]



25 -7

-8